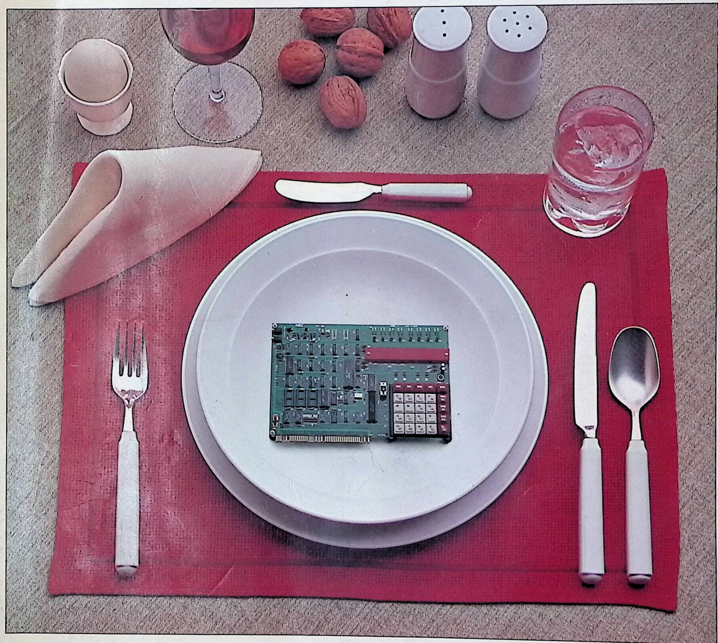
マイクロコンピュータ

TK-85

TRAINING BOOK

ハードウェアとソフトウェア



Nippon Electric Co.,Ltd.

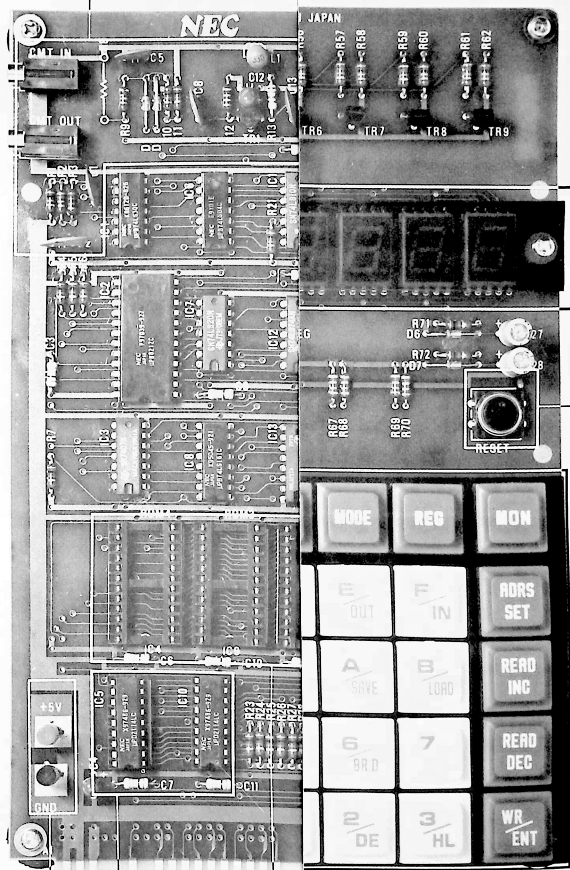
TK-8550
MAY-1980

マイクロコンピュータ

TK-85 TRAINING BOOK

ハードウエアとソフトウエア

日本電気株式会社



は改良のため予告なく変更されることがあります〕

附属品の確認と足の組み立て

TK-85 の附属品

TK-85 のケース内には TK-85 本体の他に下記に示す附属品が梱包されています.

《附属品》

○TK-85 TRAININK BOOK 1部

○TK-85 本体支持材一式（ゴム足，スペーサ，ビス類）

○電源用コード　赤黒各1本

○保証書

以上の附属品があることを確認してください.

TK-85 の足の取り付け方法

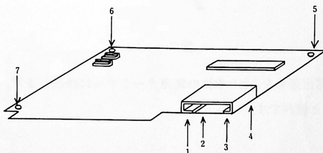
TK-85 は足を取り外した状態でケースに梱包されています. 使用する前に必ず足を取り付けてください.

ケースには足の取り付け部品として下記のものが附属品袋に入っています. 確認してください.

ゴム足	×7
スペーサ	×7
ビス（M3，25mm）	×7
ナット（M3）	×3
平ワッシャ	×3

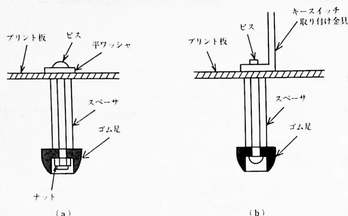
ケースより TK-85 本体を取り出したとき，キースイッチ取り付け金具は図 Aの1〜4 で示す4本のビスにより基板に仮留めされています.

この仮留めしてある4本のビスを外します.



図A

次に図Aの1~7で示す穴7箇所に図Bで示す構成どおりに基板へ足を取り付けてください.



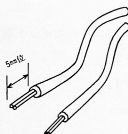
図B　取り付け方法

取り付け方法として図Aの穴1～穴4までは図B(b)の方法で,穴5～穴7までは図B(a)の方法で行ってください.

仮留めビスを外した後の取り付け作業は慎重に行ってください. 大きな振動を与えますとキースイッチのリード線が基板より外れる場合があります. 注意してください.

電源コードの接続方法

附属品の電源コードをTK-85本体に接続するには図Cのように電源コードの先端の被覆を取り除きます.



図C

被覆を取り除いた後,露出部分をよじり本体の電源ターミナルに接続します. 接続する場合赤色コードを+5Vと決めると便利です.

まえがき

トレーニング マイクロコンピュータTK-85は，マイクロコンピュータを理解し，マイクロコンピュータを利用したアプリケーションを開発される方の入門機として最適な製品です.

TK-85は広く一般に知られている当社製品TK80/80Eの流れをくむもので同一の命令で動作する一方，それ以上の機能を含み，従来よりもモニタプログラムが強化され，プログラミングやデバッグが簡単に行え，さらにハードウェアの拡張も簡単に行えるように配慮されています.

本書には，TK-85を御利用頂くための説明のほか，マイクロコンピュータの基礎に関することがわかりやすく解説されています. またTK-85を利用したプログラム例も記載されていますので，実際にTK-85を動作させながら体験的にマイクロコンピュータを学んでいくことができます.

TK-85および本書が広い分野で活用され皆様のお役に立つ事を期待いたします.

目　次

4章 プログラムの基礎と作り方

5章 データの取り扱い

6章 制 御 文

7章 サブルーチンの考え方

8章 モニタの詳細な説明

9章 モニタプログラムリスト

10章 モニタサブルーチンの使い方

11章 TK-85のハードウェア

付録

0章　マイクロコンピュータの魅力

1．はじめに

本章ではまず初めにマイクロコンピュータがどのようにして誕生したかを述べ，次にそのマイクロコンピュータがいかに利用されているかを見ていきます．そして最後に簡単にマイクロコンピュータの概念について勉強することにしましょう．

2．マイクロコンピュータの誕生まで

1948年，この年に世界最初のコンピュータENIACがアメリカのペンシルバニア大学で誕生しました．このENIACは，大きさが二十畳程の部屋一杯を占めるくらいの巨大なもので，真空管やリレーを何万個も使用してあり，数百キロワットの電力を消費するものでした．

現在のコンピュータと比べるとその性能はきわめて低いものでしたが，それでも当時の人達にとっては，機械が自動的に計算をしたり，いろいろな処理をすることができるということは大きな驚きでした．

このようにして生まれたコンピュータはその後，半導体技術と相まって進歩を重ね，今日に至っています．現在では，コンピュータの性能は非常に高く，そのため，コンピュータといえば神様のような万能の機械と思っている人さえいるほどです．

それでは，なぜコンピュータはこれほどまで進歩し続けてきたのでしょうか？　コンピュータはどのような点で人間社会に貢献してきたのでしょうか？

それはコンピュータには人間の能力より優れている点がいくつかあったからです．その人間より優れている点として次の三つの事柄をあげることができます．

(1)　計算・処理速度が人間に比べて桁違いに速い．

(2)　単純な繰り返しあるいは複雑な作業を長時間やらせても疲れないし，またミスもしない．

(3)　一度記憶したことは絶対に忘れない．

これらのコンピュータの長所は，人間が数多くの情報を処理したり，複雑な計算をしたり，たくさんのことを記憶しておいたりするには大変都合のよいものだったのです．大量の情報を処理したり，複雑な計算を人間がやっていたのでは時間がかかるし，ミスをするかもしれません．また，記憶しておくべきことを忘れてしまうこともあります．

コンピュータを使用すればこれらの仕事を正確に速くやってくれます．このコンピュータの長所をより強力に生かすために，コンピュータは進歩し続けてきたのです．

さてこのように進歩してきたコンピュータの歴史の中で，我々がこれから見ていこうとするマイクロコンピュータはどのようにして生まれてきたのでしょうか．

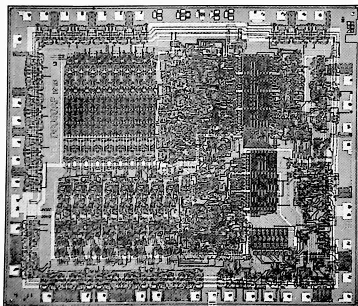
エレクトロニクスの進歩は目を見張るものがあります．より高性能かつ小型で低消費電力のものを求めて真空管からトランジスタへ，そしてトランジスタからIC(集積回路)へ，さらにはLSI(大規模集積回路）へと飛躍的に進歩を続けています．現在の最新鋭のLSIはわずか数ミリメートル四方のシリコン基板上にトランジスタの数にして数万個を詰め込んだものが作られ，高性能・小型・低消費電力，そしてある程度の数を作れば非常に安価なものが実現されています．

このようなLSIは多くの分野で使用され，現在ではLSIはエレクトロニクスの大きな柱となっています．しかし，各分野で使うLSIはそれぞれ全く違った仕様が要求されます．ですから各分野からの要求に応えるためにはそれぞれ違った仕様のLSIを何種類も作らなくてはなりません．しかし，LSIを作るのは非常に面倒で，一つのLSIを新しく作るには長い時間と多くの人手がかかります．しかし，LSIはひとたび作りだすと，同じ仕様のものを比較的簡単に大量生産することができるという長所があります．そこで次のような考え方が生まれました．

「利用者から注文があるたびに新しいLSIを作るのではなくて，どうにかしてどんな分野にでも使えるLSIを作ることはできないものか？」

これに対する解決策はLSIにコンピュータの機能をもたせることでした．コンピュータはプログラムを変えれば，どのようにも利用できることからLSIにコンピュータの機能をもたせ，あとはプログラムを変えるだけで，同一LSIであらゆる分野の要求に応じようという考えです．

つまりそれまでのLSIを印刷された紙と考えるなら，コンピュータの機能を持たせたLSIは真白の紙であり，利用者がそれぞれ自由に書きこめばよいようにしたのです．



LSIの内部拡大写真 (μPD8080AF)

もうおわかりになったでしょう．そうです，コンピュータ化されたLSI，またはLSI化されたコンピュータといってもよいのですが，それがマイクロコンピュータなのです．

このアイデアを実現したものが1971年，アメリカのINTEL社より発表された4004というマイクロプロセッサで，今日のマイクロコンピュータブームを生み出すもととなりました．

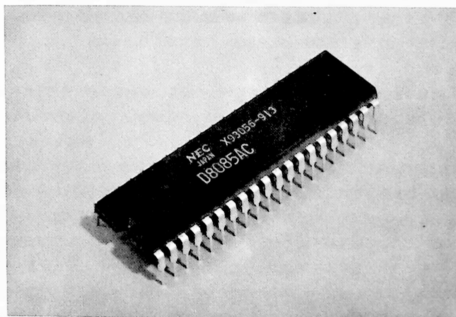
3．　マイクロコンピュータの利点，欠点

2節で述べたようにして生まれたマイクロコンピュータはそれまでのコンピュータの利点とLSIの利点の両方をかねあわせて持っています．つまりコンピュータの長所である処理速度が速く，正確でミスをしないなどに加えて，コンピュータが小さく，軽く，安くなったのです．

世界最初のコンピュータENIAC以後，コンピュータの性能は大きく進歩していき，そして多くの分野の人々に使われてきましたが，絶対数からすれば非常に少数の人々にしか利用できませんでした．なぜでしょうか？　それは，コンピュータは高価なものであり，専門の人以外には容易に利用できるものではなく，形も大きなものだったからです．ですからコンピュータを利用したくても価格の点や大きさの点などで使用することのできない人が多かったのです．

ところが，そこへマイクロコンピュータが登場したのです．それまでのコンピュータに比べて価格は大変安く，小さく，また性能のほうもそれまでのコンピュータに比べてそう劣ってはいないのです．これが世間の注目をひかないわけがありません．

実際，マイクロコンピュータはたちまちのうちに多くの人々の，それも今度は絶対数にしても多くの人々から興味をもたれたのです．



マイクロコンピュータCPU

本節の題に「マイクロコンピュータの利点・欠点」と書きましたが，マイクロコンピュータは大容量・超高速処理の要求を除けば，それまでのコンピュータに見られる多くの使用上の制約がそのまま利点となり，利点は数多いけれども欠点はなかなか見つからないほど素晴らしいものになっているのです．

今まで，コンピュータの使用を考えたときにあった障害のほとんどを取り除いたのですから，これからはいろいろな分野にマイクロコンピュータの使用を考えることができるのです．その応用範囲は無限といっていいぐらい広がったといえます．もうコンピュータは特殊なものではなくなったのです．

これはある一部の人達にしか使用できなかったコンピュータがあらゆる分野の人達にも使えるように道が開けたという点でコンピュータの歴史の中での一大エポックなのです．

読者のみなさんもマイクロコンピュータの知識を得られたなら，マイクロコンピュータを活用する新しい分野をみつけ，そしてそれを開拓していっていただきたいと思います．

4．　マイクロコンピュータの応用

マイクロコンピュータは安価で小型軽量，低消費電力など数々の長所を生かしていろいろな分野に活用することができると書きましたが，具体的には今後どのような使われかたをしていくのかを考えてみましょう．

文明の進歩は我々の生活を次第に豊かにしていきました．我々は機械を使うことを覚え，その機械を使う便利さを得てきました．

しかしこのような社会において，我々は機械を操ってはいますが，逆に機械に操られていることも少なくないのです．そのような悲劇をあつかった映画にチャップリンの「モダン・タイムス」がありましたが，我々は機械を100％操ってはいないのです．

たとえば自動車について考えてみましょう．

我々が本当に自動車を奴隷のように100％操っているのなら，自動車に向かって「オイ，駅へ行け!!」などと命令すれば自動車が自動的に命令されたとおりに駅へ行くのが本当ではないでしょうか？

現実はどうでしょう．自動車を動かすにあたって「ガソリンははいっているか？」とか，運転しているときには「事故をおこさないようにしなければ」などと思いながらハンドルを動かし，目を常に前後左右にひからせているではありませんか！

勿論，なかには「そのように運転するのが楽しいんだ」という人もいるかもしれませんが，それは別として，一般に我々は自動車を奴隷のようにはあつかっていないことがわかるでしょう．

それでは自動車に「行け!!」と言っただけで自動車が進むにはどうすればよいのでしょうか．

いちばん良いのが自動車自身にロボット，つまり機械人間になってもらい，自動車が自分で「ガソリンはあるか」とか「左右に人はいないか」などを判断して動くようになればいいのです．

少し話が現実味をもたないようですが，たとえばロケットはコンピュータによって操縦されていることや，飛行機なども離陸から着陸までコンピュータにまかせっきりにできる現実を考えるなら，そう現実味のない話ではありません．

話をもっと身近なものにしましょう．

あなたの家にクーラーがあるとします．あなたはそのクーラーを使用するにあたって「何時になったらクーラーを作動させ，室温が○○度になったら，その後その温度を常に一定に保つように動作する」ようにしたかったとします．

もし，クーラーが機械人間になってくれたなら……．今度の話はもう実用化されているものです．

ではこの機械人間はどのようにすれば可能になるのでしょう．そうです．このような場合にマイクロコンピュータを使用するのです．

今までのコンピュータを使用しても機械人間を作ることは可能ですが，コストの点や，クーラーより大きなコンピュータを別に設置しなければならないなど非現実的なものとなります．マイクロコンピュータは小さいものですから，クーラーの中に組み込めます．また，価格が安いからクーラー自身の価格にもそうひびきません．

マイクロコンピュータはこのようにいろいろな機器に頭脳を持たせて，人間の仕事を肩代わりさせるための強力な武器となります．

もちろん今までコンピュータで行っていた仕事の一部をマイクロコンピュータに肩代りさせることもできます．しかし，マイクロコンピュータは，マイクロコンピュータにしかできない応用に用いたとき，その効果が如何なく発揮されます．その一つの例が今まで述べてきたいろいろな機械を，外観はほとんど変化させずに機械人間にすることなのです．

近い将来，あらゆる機器が機械人間になり，人間がどんな機械も100％操れるようになるのではないでしょうか．

ではこのような考え方にそって，実際，マイクロコンピュータは現在どのように活用されているのかを紹介しましょう．

4.1　自動車への応用

先程自動車の究極的なかたちまでを考えましたが，現実はまだまだ夢物語です．逆にいえば，自動車という機械は人間に大変多くの動作を要求してくる機械なのです．ですから，自動車にはコンピュータを導入する部分がたくさんあるのです．

現在，自動車にマイクロコンピュータが導入されている部分は主にエンジン関係やブレーキ関係などです．

近年，深刻な石油不足から，自動車に対して，低燃費，つまり，より少ないガソリンでより長い距離を走る自動車が求められています．

このような状況下において，各自動車メーカーはエンジンをいつかなるときにも最も効率よ

く働かせるように努力しています．しかし，エンジンの機械的部分そのものは飛行機の技術などとあいまってきわめて高い水準に達してしまっているため，機械的部分の改良の余地はほとんどないくらいになっています．そこで，逆にエンジンをいかなる状態でも最も効率よく働くようにコントロールすることを考えました．

たとえば，エンジンの最適点火時期を決定させること．あるいはエンジンが最も良く働くとされる空気・燃料比（14.7対1といわれる）を常に保つために吸入空気量と排ガス中の酸素の比を測定して，燃料噴射装置をコントロールするなどです．

ブレーキ関係では，走行中の自動車が停止するには，ブレーキペダルを踏むことによって油圧を上昇させそれがホイールに伝わり，その結果タイヤと路面の摩擦によって止まります．そのとき，あまり強くブレーキペダルを踏むとホイールがロックされてスリップしてしまい，制動距離がかえって長くなってしまったり，路面の状態次第では，操縦不能になることさえあります．ブレーキがいちばん良くきくのはホイールがロックされる一歩手前の状態を保つときです．これを実現する装置をアンチスキッドブレーキといいます．これはホイールのロックを検出したら油圧を自動的に下げ，ホイールのロックを解除する機構になっています．

以上，エンジンとブレーキの二つの例をあげましたが，この二つとも制御部分にそれぞれマイクロコンピュータが使われています．これらの部分はマイクロコンピュータが出現しなければ全く不可能か，または大変困難なことだったのです．

このようにして自動車は現在，着々と完全な機械人間に向けて進歩しています．近い将来，完全な機械人間となった自動車ができることでしょう．

4.2　ゲーム機器への応用

インベーダーブームというのがありました．御存知だと思いますが，あのテレビゲームのインベーダーゲームのことです．みなさんの中にも目を血走らせながら100円玉を湯水のごとく使って，インベーダーゲームにしがみついていた時期があったかもしれません．このインベーダーゲームをはじめ，最近に見られるテレビゲームの類のほとんどにマイクロコンピュータが使用されています．この応用例は我々の生活を豊かにする機器の自動化・省力化とはちょっと異なる点がありますが，我々に従来と違った娯楽をあたえてくれるという応用では大変身近なものですから，マイクロコンピュータの応用例として少し書いてみましょう．

ブロックくずしというゲームを御存知ですか？　そうです，「ピポッ・ピポッ」といいながら落ちてくる玉を下方にあるパドルではね返し，上方にあるブロックを消していくゲームです．さて，このブロックくずしというゲームは，画面などを見ればわかるようにインベーダーゲームに比べて，比較的単純な処理しか必要としません．しかし，このブロックくずしでさえ，もしLSIを使用せずに作るとなるとその電子回路は大変複雑なものになり，その結果，人手と時間がかかり，コストアップを招き実用化（つまりゲームなどにはそうたくさんの金はかけられないから）はむずかしいといえます．

だからインベーダーゲームをはじめとするテレビゲームなどではLSIを使わずに作ることは非現実的なことといえます．そのLSIとして，マイクロコンピュータが使用されているのです．もし普通のLSIをつかうと，新しいゲームを開発するたびに新しいLSIを開発しなければなりません．

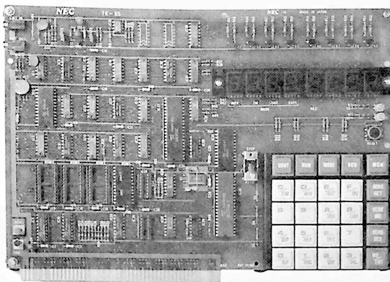
マイクロコンピュータには，コンピュータの特徴であるプログラム次第でいろいろな働き方をするという長所があります．このことはプログラムを開発することでいろいろな種類のLSIを作り出すことができるといえます．しかもプログラムの開発は人間の頭脳によるだけで多大な設備はいりません．ですから，マイクロコンピュータの知識をしっかり持っていれば，新しいゲームの開発が大変らくになりますし，また自分でいろいろなゲームを作り出すこともできます．

現在，このようにして新しいゲーム機器が次々と開発され，ゲーム場に送り込まれています．あなたがそのゲーム機器でゲームを楽しんでいるとき，それはとりもなおさず，あなたはマイクロコンピュータを操っているのです．

4.3 パーソナルコンピュータへの応用

今までのコンピュータは高価で形も大きく，大変あつかいづらいなどから，個人が自分専用のコンピュータを所有することは本当に夢物語でした．

そこへマイクロコンピュータが登場しました．マイクロコンピュータは，それまでのコンピュータでは考えられなかった応用が開けてきたのと同時に，それまでのコンピュータの性能をある程度保ったまま大変安価なコンピュータを供給できるようになったのです．そしてその値段は個人で買える値段のものさえでてきました．そこで，個人専用のコンピュータという考え方，すなわちパーソナルコンピュータという考え方がでてきたのです．



TK-85外観

さて，このコンピュータを個人所有していったいなにをするのでしょうか？

自分だけの情報，記憶，管理すべきことをこのパーソナルコンピュータに肩代りしてもらうこともできるでしょう．

教育機械としての応用，つまりコンピュータをうまくプログラムすることによって教科書の代りや先生の代りをさせることもできるでしょう．あるいは趣味としている人もいるかもしれません．

また，パーソナルコンピュータを使ってマイクロコンピュータの勉強をするということもできます．

このパーソナルコンピュータはマイクロコンピュータの応用分野として新しく開拓されたので，今後，個人用として，また家庭内において，いろいろな利用方法が考えられます．

以上三つのマイクロコンピュータの応用例を見てきましたが，マイクロコンピュータの応用例はこれだけにとどまっていません．ちょっと思いうかべただけでも，カメラ，ミシン，オーディオ機器，クーラー，テレビ，計測器，印刷システム，時計，シンセサイザ，教育用機器，アニメーション，おもちゃ，自動販売機，工場における制御機器，大型コンピュータの端末…… etc. とあります．

このように，マイクロコンピュータの応用分野は無限に広がっていき，もはやコンピュータはひとつのパーツとして扱われる時代を迎えています．

5.　マイクロコンピュータの基礎

今まで，マイクロコンピュータの生い立ちやその応用についてお話ししてきました．次章からマイクロコンピュータを実際に使って勉強していただくわけですが，その前にマイクロコンピュータを勉強していく上での基礎知識としてマイクロコンピュータの概念を述べておきます．

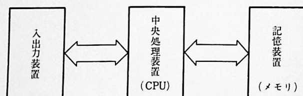


図0.1

図0.1を見て下さい．マイクロコンピュータは大きくわけて三つのブロックに分かれています．それぞれの三つのブロックを簡単に説明してみましょう．

①　入出力装置とはマイクロコンピュータの外部からの信号をマイクロコンピュータ内部へ受

け入れる役割りをもっています．また逆にマイクロコンピュータの内部の信号を外部に出す役割りもします．人間でいうと目，耳や口にそれぞれあたります．

② 中央処理装置とは入力装置から入ってきた信号や記憶装置内にある内容によって実際にいろいろな仕事をするところであり，コンピュータの中心をなすとこです．

③ 記憶装置はその名のとおり，いろいろな情報を記憶するところです．その情報は中央処理装置の命令によって記憶されることもあれば，その記憶した内容が呼びだされることもあります．

中央処理装置と記憶装置は人間でいえば頭にあたるでしょう．

なお中央処理装置のことを英語でCPU（Central Processing Unitの略です）といい，この言葉が使われることが多いので，以後中央処理装置のことをCPUと呼ぶことにします．同様に記憶装置のことはメモリ(Memory)と呼ぶことにします．

マイクロコンピュータは以上述べた部分から構成されています．しかし，マイクロコンピュータは電源を入れただけでは，動作しません．マイクロコンピュータは，最初なにをすればよいのか知らないのです．それでは最初どのようにして動作させるのでしょうか．

マイクロコンピュータにかぎらずコンピュータと名のつくものは次の二つの条件をもっています．

(1) メモリの中に処理の手順を書いたプログラムを持っていること．

(2) 自動的にそれらの処理を実行することができること．

上記の二つの条件はどのような意味なのでしょうか？

先程CPUはいろいろの処理を実行する部分であるといいましたが，CPUそれ自体でできることはごく基本的なことだけなのです．それにもかかわらず複雑で難しい処理をコンピュータができるのは，ごく基本的な命令を巧みに組み合わせておくことによってコンピュータは見かけ上複雑で難しい処理を行っているのです．

今までプログラムという言葉を何回か使ってきましたが，この基本的な命令を巧みに組み合わせ，コンピュータを見かけ上複雑な処理ができるようにすることをプログラムするといいます．

このようにコンピュータは，プログラムを与えることによって始めて動作します．そして，このプログラムを一度実行させたら，あとはコンピュータはプログラムに書いてあるとおりの手順を自動的に実行していくのです．

さらにコンピュータを詳しく見ていってみましょう．CPUはどの様にしてメモリからプログラムを読み出し，命令の実行をしていくのでしょうか？

プログラムというものはプログラムが書かれた段階ではそれ全体としてどんな仕事をするにせよ，ある基本的な命令が順番にならんでいます．

そのようになっているプログラムをメモリに記憶させておいて，それぞれの命令をCPUがメモリから呼びだしCPUが実行するのです．そのときCPUが，それぞれの命令のならんでいる順番を知るめやすとして，それぞれの命令にプログラムに書いてある順番に番地(Address：アド

レス）をつけておくのです．たとえばメモリの1番地にある命令，2番地に次の命令，以下3番地，4番地，……というように続いていくのです．そしてCPUはプログラムの実行時にはその番地を順に読んで，そこに書かれている命令を順に実行するという方法をとっているのです．

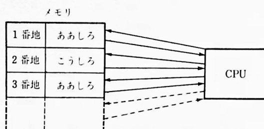


図0.2

メモリの基本的な働きはある情報を自由に記憶したり，あるいは思いだすことです．メモリに情報を記憶させることを書き込み（Write），思いだすことを読み出し（Read）といいます．さてマイクロコンピュータに使われているメモリにはどのようなものがあるでしょうか？

マイクロコンピュータに使われているメモリは大きくわけて二つあります．その一つをRAM（Random Access Memory：ラム）といいます．マイクロコンピュータに使われているRAMは半導体回路でできているのですが，普通，半導体で作られたRAMは電源を切ると記憶内容はメチャメチャになってしまいます．しかしRAMは読み出し，書き込みとも自由にできる便利なメモリです．他の一つにROM（Read Only Memory：ロム）というメモリがあります．これはRAMと違い電源を切ってもその内容は保存されますが，読み出しだけにしか使えないメモリです．RAMもROMも別にどちらが良いというわけでなくどちらも一長一短があり，場合に応じて使いわける必要があります．

この章ではマイクロコンピュータの生いたちからその応用例，そしてマイクロコンピュータの基礎についてお話ししてきました．では次章から，実際にマイクロコンピュータの世界に足をふみ入れていきましょう．

1章　マイクロコンピュータを動かしてみる

1．　はじめに

0章のおわりでマイクロコンピュータの基礎知識についてお話をしましたが，まだほかにも知らなければならないこともたくさんあります．しかし習うより慣れろという諺もあります．詳しい知識の説明をするまえに TK-85 を使って，とにかく自分の手でマイクロコンピュータを動かしてみましょう．

2．　TK-85 を使う前に

まず TK-85 を使う前にそれを動くようにセットしなければなりません．

TK-85 を動かすには電源として＋5Vの電源を用意し，TK-85 についている＋5V端子と電源の＋5V端子，およびTK-85 の GND（アース）端子と電源の GND 端子を，それぞれ間違いないようにしっかりとつないで下さい（図1．1参照）．

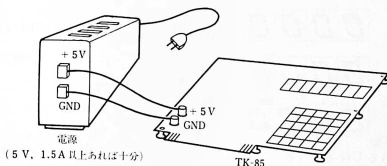


図1．1

次に電源を投入する前に TK-85 の各部分の簡単な説明をしておきます．TK-85 の右下のほうに25個の押しボタン（キー）が並んでいます．この部分をキーボードといい，キーを押すことにより TK-85 にプログラムを入力したりいろいろな命令を TK-85 に入力します．このキーボードの上に数字などを表示する赤く点燈する部分があります．この部分のことを LED ディスプレイ（以下，単に LED と記します）といいます．この LED が人間に対する表示機能です．これら，キーボードおよび LED が0章で述べたコンピュータの入出力装置に当ります．そしてほかのいろいろな半導体や抵抗，コンデンサなどが配置されているところが CPU およびメモリの部分と考えてください．

3．　TK-85 を動かしてみる

それでは，電源を入れましょう．電源を入れたら次に RESET と書かれたボタンを押します．LED は次のように表示するはずです．



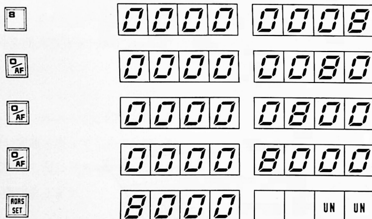
もしこのように表示されなかったり，電源を入れたあと TK-85 から煙やへんな臭いなどがしたらすぐに電源を切って TK-85 と電源との配線に間違いがないかを見て下さい．

LED が上記のようになったら TK-85 は正常に動いています．

つづけて次の順にキーを押していってみてください．



以下に上記のように押したときの LED の表示を示して見ます．

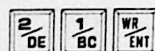


上記の最後の行にある UN という表示はなんらかの文字で表示されますが，それが一定していません．UN としてある箇所は何か文字あるいは数字が表示されれば，どんな表示であっても気にすることはありません．

先を続けましょう．いままでの操作で LED は次のような表示になっています．



つづけて，次の順にキーを押します．



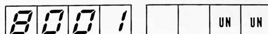
以下にそのときの LED の表示を示します．

押すキー　　　　押したあとの LED の表示







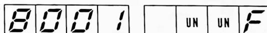
いまやっていることを説明しておきます．この LED の左 4 桁の表示はメモリの番地を示しています．LED の右の 4 桁はそのメモリの番地に書き込まれる内容です．つまり今，メモリの8000番地に21というデータ（情報）を書き込んだのです．

ただし，この8000番地の8000やデータの21は我々の普段使う10進数の8000や21ではなく，16進数の8000や21です．そのため，0～9のほかにＡｂＣｄＥＦなどという文字が0～9の数字にまじっていることがあります．16進数の話は2章でしますので，とにかくこの8000や21は我々が普段使用している10進数の数字とは少し違っていることを覚えておいて下さい．

ここでキーを押し間違えたときのことを説明しておきます．たとえば [F/IN][8] と押すべきところを [F/IN][7] と押してしまったら，[WR/ENT] を押す前に続けて [F/IN][8] と正しく押し直した後 [WR/ENT] を押せばいいのです．こうすれば間違わずにキーを押したときと同じように TK-85 は理解してくれます．

この過程を LED の表示とともに示しておきます．

キーを押す順序　　　　そのときの LED の表示

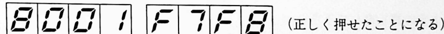





（間違えた）



（押し直し）



（正しく押せたことになる）




また全然ちがうキーを押してしまったら，[RESET] ボタンを押して最初からやり直して下さい．

話を続けます．次のようにキーを押していって下さい．いままでの操作で LED は下のように表示しています．

キーを押す順	そのときの LED 表示	
	8001	_ _ UN UN
[F/IN]	8001	_ UN UN F
[8]	8001	UN UN F 8
[WR/ENT]	8002	_ _ UN UN
[8]	8002	_ UN UN 8
[3/HL]	8002	UN UN 8 3
[WR/ENT]	8003	_ _ UN UN
[3/HL]	8003	_ UN UN 3
[E/OUT]	8003	UN UN 3 E
[WR/ENT]	8004	_ _ UN UN
[0/AF]	8004	_ UN UN 0
[8]	8004	UN UN 0 8
[WR/ENT]	8005	_ _ UN UN

以下，LED の表示は略してキーを押す順番だけを書きます．キーを押し間違えたときは前述のように押し直して下さい．

[3/HL] [6/BRD] [WR/ENT] [4/SP] [0/AF] [WR/ENT] [C/TM] [D/MOV] [WR/ENT] [3/HL] [0/AF] [WR/ENT] [8] [0/AF] [WR/ENT] [3/HL] [6/BRD] [WR/ENT] [0/AF] [0/AF]

[WR/ENT] [2/DE] [3/HL] [WR/ENT] [3/HL] [D/MOV] [WR/ENT] [C/TM] [2/DE] [WR/ENT] [0/AF] [5/BRP] [WR/ENT] [8] [0/AF] [WR/ENT] [C/TM] [3/HL] [WR/ENT] [0/AF]

[0/AF] [WR/ENT] [8] [0/AF] [WR/ENT]

ここで [8] [0/AF] [3/HL] [0/AF] [ADRS/SET] と押します．LED の表示は

8030　UN UN

となります．そして次の順にキーを押し続けます．



以上でキー入力は一応終りですが，もしかしたら気がつかないうちに押し間違いをしているかもしれません．そこでチェックをしてみます．

以下，正しく押されたときの状態をお見せしますから，あなたの TK-85 の LED の表示と比べてみて，もし表示と違っていたら直します．その直しかたはあとで説明しますから，とにかくあなたの LED の表示と比べて，違ったところがあったらその箇所をなにかにメモしておいて下さい．

[8] [0/AF] [0/AF] [0/AF] と押します．

以下，押したキーと LED の表示は次のようになります．

押したキー	LED 表示	
ADRS SET	8000	21
READ INC	8001	F8
READ INC	8002	83
READ INC	8003	3E
READ INC	8004	08
READ INC	8005	36
READ INC	8006	40
READ INC	8007	Cd
READ INC	8008	30
READ INC	8009	80

READ INC	800A	36
READ INC	800b	00
READ INC	800C	23
READ INC	800d	3d
READ INC	800E	C2
READ INC	800F	05
READ INC	8010	80
READ INC	8011	C3
READ INC	8012	00
READ INC	8013	80

次に [8] [0 AF] [3 HL] [0 AF] と押します．

ADRS SET	8030	16
READ INC	8031	40
READ INC	8032	06
READ INC	8033	00
READ INC	8034	05
READ INC	8035	C2
READ INC	8036	34

押すキー	LED 表示
[READ INC]	8037　　80
[READ INC]	8038　　15
[READ INC]	8039　　C2
[READ INC]	803A　　32
[READ INC]	803b　　80
[READ INC]	803C　　C9

さて，以上を見てきて，間違がひとつもなければ結構です．しかし，もし間違いがあったら次のようにして直しておいて下さい．

間違っていたところはなにかにメモしてあるはずですね．たとえば，

8011　　C3

と表示されるところが

8011　　C2

と表示されたとします．

[8] [0 AF] [1 BC] [1 BC] [ADRS SET] と押しますと LED に

8011　　C2

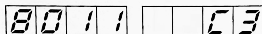
と表示されます．続けて [C TM] [3 HL] [WR ENT] と押します．これで間違いは直りました．以下にそのときのようすを示します．

押すキー	LED 表示
現在	8011　　C2
[C TM]	8011　C2C
[3 HL]	8011　C2C3
[WR ENT]	8012　　00

これで



を



に変えることができました．もし間違った箇所があったらこの例に従って直して下さい．

さあ，これでチェックもおわりました．

もうわかっている人もいるかもしれませんが，いままでやってきたことは TK-85 のメモリにプログラムを書き込んでいたのです．そしてそのプログラムをメモリに書き込む仕事はおわりました．

では TK-85 にプログラムを実行させてみましょう．

次の順にキーを押します．



これで TK-85 がプログラムを実行します．

いかがですか？　LED の上を左から右へ矢印が次々に動いていったと思います．もしそうならなかったら先程のチェックが完全ではなかったのです．もう一度，前の手順に従ってチェックを行って下さい．

なかなか面白いプログラムでしょう．しかしいつまでも見ているとあきてきます．この矢印プログラムの実行をやめるには MON を押して下さい．プログラムの実行を止めたときのアドレスとアキュムレータとフラグの内容が LED に表示されます．その後また動かしたかったら CONT を押せばよいのです．

この矢印プログラムが，どのようにしてコンピュータによって実行されているかは次章以後で詳しく説明します．ここではこの矢印プログラムとはどのように書かれているかをちょっとお見せしておきます．

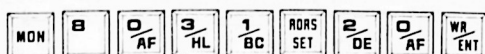
アドレス	ニーモニックコード	オペランド	マシンコード
8000	LXI	H,83F8H	21F883
8003	MVI	A,08	3E08
8005	MVI	M,40H	3640
8007	CALL	8030H	CD3080
800A	MVI	M,00	3600
800C	INX	H	23
800D	DCR	A	3D
800E	JNZ	8005H	C20580
8011	JMP	8000H	C30080
8030	MVI	D,40H	1640
8032	MVI	B,00	0600
8034	DCR	B	05
8035	JNZ	8034H	C23480
8038	DCR	D	15
8039	JNZ	8032H	C23280
803C	RET		C9

リスト　1.1

これを見てもおそらくチンプンカンプンでしょう．でも心配することはありません．本書を読んでいくうちに，すらすらと理解できるようになりますから．

次にこのプログラムをちょっと改造してみます．

次のようにキーを押します．



押し間違えたらもう一度最初からキーを押して下さい．これでプログラムの改造はおわりました．どのように改造されたのか，実行させてみましょう．

先程の手順と同じく，8 0 0 0 ADRS SET RUN です．

矢印のスピードが速くなったでしょう．今プログラムを改造するときに押した 2 0 の部分を違う二つの数字に変えてみるとまた矢印のスピードが変ります．試してみて下さい．

もうひとつやってみます．



またプログラムを改造しました．実行させてみましょう．矢印に代って8が左から右へ移動するようになったでしょう．プログラムはこのように少し変えるだけでずいぶん違った結果がでてくるでしょう．

このことは覚えておいて下さい．

4．　テープレコーダとの接続

これまでTK-85にプログラムを書き，それを実行してみたわけですが，TK-85の場合，電源を切るとせっかく入れたプログラムをすべて忘れてしまいます．ですから一度電源を切ってしまってからもう一度矢印プログラムを実行してみたいと思ったら，またこの章の最初からやってきたことを繰り返さなければならないこととなり，大変不便です．TK-85ではこのような不便をなくすために入力したプログラムをテープレコーダに記憶させておくことができるようになっているのです．

では実際にテープレコーダにプログラムを保存させてみましょう．

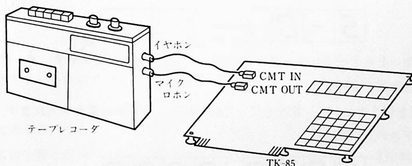


図1.2

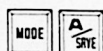
まずあなたの手近にあるテープレコーダを用意してください．それと TK-85 を接続します．

TK-85 の IN と書いてあるジャックとテープレコーダのイヤホンジャックを，そして TK-85 の OUT と書いてあるジャックとテープレコーダのマイクロホン入力ジャックをそれぞれつなぎます（図1.2参照）．

これで TK-85 とテープレコーダの接続は完了です．テープレコーダにテープを入れます．

テープを完全に巻き戻し，テープのリーダ部分が巻き込まれるまでテープを少し巻いておきます．

それから，次のようにキーを押します．

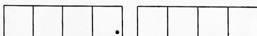


すると LED は次のように表示します．



[0/AF] [1/BC] [WR/ENT] とキーを押します．

LED は再び



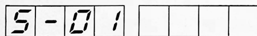
となります．

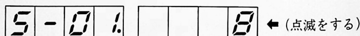
それから [8] [0/AF] [0/AF] [0/AF] [WR/ENT] [8] [0/AF] [3/HL] [F/IN] と入力すると LED に



と表示されます．テープレコーダを録音状態にしてスタートさせます．そうしてから，

[WR/ENT] を押すと以後 LED は以下のように変化します．







最後の状態になったらテープレコーダにプログラムの書き込みは完了です．テープレコーダを止めてテープを巻き戻しましょう．

TK-85 の電源を切りましょう．電源を切って10秒程たったらまた電源を入れます．

[RESET] を押して下さい．LED は



になっていますね．

ではTK-85は電源を切るとプログラムを忘れてしまうことの証明に，前述のプログラムが入っていた状態と同じ操作をしてみましょう．



どうです．矢印が左から右へ流れましたか？　流れなかったでしょう．TK-85は電源を切るとプログラムを忘れてしまうのです．また矢印プログラムをTK-85に実行させたい場合は，またこの章の始めからの手順でやり直しますか？そんなことをしなくても大丈夫です．このためにTK-85とテープレコーダを接続して矢印プログラムをテープに記録したのです．ではさっそくテープレコーダからTK-85にプログラムを入力してみましょう．

テープレコーダの，ボリュームつまみを最大と最小の中間ぐらいにおきます．記録したテープを完全に巻き戻しておいて[MODE] [B/LOAD]と押します．LEDは，

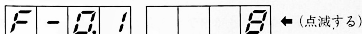


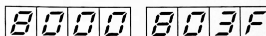
となります．

[0/AF] [1/BC] [WR/ENT]と押すとLEDは



となります．テープレコーダをスタートさせます．そのうちLEDが



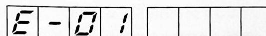


となったらプログラムはうまく入力されました．



と押せば矢印プログラムが動くでしょう．

問題はLEDが，



となったときです．これはテープレコーダからTK-85へプログラムがうまく入力できないことを表します．

このように表示されたらテープを巻き戻して，[RESET] [MODE] [B/LOAD] [0/AF] [1/BC]と押してテープレコーダのボリュームの位置を適当に変えて[WR/ENT]を押しプログラムのロードを2～3回試みて下さい．プログラムをTK-85にうまく入力できたなら，今後，TK-85と共に使うときは，そのテープレコ

ーダのボリュームの位置をそのうまくいったときの位置にして使って下さい．

2～3回試みて，うまく入力されないときは手数がかかりますがこの章の最初からの手順でやり直して下さい．テープレコーダの故障か接続ミスがないかぎり TK-85 にプログラムがうまく入力できないことはほとんどありません．しかし，何度やってもだめなときは TK-85 を買ったお店などに相談してみてください．

この章では TK-85 をとにかく動かしてみるということを目標にしてきました．あなたの TK-85 はうまく動きましたか？

しかし，あなたが TK-85 を動かせたといってもその原理を十分理解して動かしたのではありません．次章からは TK-85 はどうやって動いているのか，その中をのぞいてみましょう．

2章　マイクロコンピュータの中をのぞく

1．はじめに

マイクロコンピュータを自由自在に使用するために知らなければならないことがたくさんあります．その知識は単に電気・電子工学の知識だけではなく，数学の知識なども必要です．

この章ではこれらの知識の中から特に重要な知識をセレクトして，それらをやさしく述べてみることにします．

2．2進数と16進数の話

コンピュータは2進数しか扱うことができないということを聞いたことはありませんか？　そのとおりなのです．それはマイクロコンピュータでも同様です．では2進数とはいったいどのような数でしょう．

我々は日常，10進法で表される10進数を用いています．しかし，日常使われている記数法には10進法以外のものもあります．たとえば，12個で1ダース，12ダースで1グロス(12進法)，また60秒で1分，60分で1時間（60進法）などがあります．

これらはすべてその桁上がりに違いがあります．2で桁が上がる数，それが2進数なのです．

2進数は2で桁が上がるので

0，1

の二つの文字しか必要ありません．このことは10進数に比べ計算が非常に簡単になります．

マイクロコンピュータの電気回路では，2進数を表すために異なった二つの電圧で実現しています．たとえば電圧5Vを1（信号がある），0Vを0（信号がない）というように対応させているのです．このように異なった二つの電圧の信号のみで2進数が表現できるということは，コンピュータの回路構成が簡単になります．もし10進数でコンピュータを実現すると異なった電圧が10種類も必要となり，回路構成が複雑になり信頼性も低下します．

それではここで2進数について，その計算方法を具体的に説明しておきましょう．

2進数の足し算に使われる法則は次の三つだけです．

0＋0＝0

0＋1＝1＋0＝1

1＋1＝10

注意しておきますが最後の10は「ジュウ」と呼ばずに「イチ，ゼロ」と読みます．すなわち

１０のはじめの１は桁が上ったことを示します．10進数の１０と２進数の10は違うことに気をつけて下さい．またこれ以後10進数と２進数を区別するために，２進数の表現には10011_2のように右下に小さく$_2$と書くことにします．

２進数の掛け算の法則も次の三つです．

$0_2 \times 0_2 = 0_2$

$0_2 \times 1_2 = 1_2 \times 0_2 = 0_2$

$1_2 \times 1_2 = 1_2$

２進数の引き算，割り算の法則はどんなものでしょうか？　上の例を参考にして考えてみて下さい．

図２．１に２進数の足し算，引き算，掛け算，割り算の例をひとつずつあげておきましょう．

```
     1 0 1 1
   + 1 0 0 1
   ---------
   1 0 1 0 0  (答え)
   桁上げ 桁上げ

   足し算
```

```
   1 0 0 1
 -     1 0 1
 -----------
       1 0 0  (答え)
   桁借り

   引き算
```

```
         1 1 1
   ×     1 0 1
   -----------
         1 1 1
     1 1 1
   -----------
   1 0 0 0 1 1  (答え)
   桁上げ

   掛け算
```

```
               1 1  (答え)
 1 0 0 ) 1 1 0 1
         1 0 0
         -------
           1 0 1
           1 0 0
         -------
               1  (余り)

   割り算
```

図2.1

実際のマイクロコンピュータでは上記の加，減，乗，除の算術演算（四則演算）はすべて加算で実現しています．すなわち減算は補数（後で説明します）の加算，乗算，除算は加減算の繰り返しで行なっています．

この他にマイクロコンピュータの演算で必要なものに論理演算があります．

その代表的なものをあげると次のようなものがあります．

(1)　論理和（OR）（∨）：いずれか一方でも１のとき結果が１となる．

$1 \vee 1 = 1$，$1 \vee 0 = 1$，$0 \vee 1 = 1$，$0 \vee 0 = 0$

(2)　論理積（AND）（・）：両方が１のときのみ結果が１となる．

$1 \cdot 1 = 1$，$1 \cdot 0 = 0$，$0 \cdot 1 = 0$，$0 \cdot 0 = 0$

(3)　排他的論理和（XOR）（⊕）：片方が０で他方が１のときのみに結果が１となる．

$1 \oplus 1 = 0$，$1 \oplus 0 = 1$，$0 \oplus 1 = 1$，$0 \oplus 0 = 0$

つぎに，2進数を10進数に変える方法を説明します．

10進数で187という数字を分解してみると次のようになります．

$187=100+80+7=1\times10^2+8\times10^1+7\times10^0$

これと全く同様な考え方をすれば2進数は次のように10進数に変換できます．

たとえば1011_2は

$1011_2=1000_2+000_2+10_2+1_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0$

$=8+0+2+1=11$

となります．

1001_2は

$1001_2=1\times2^3+0\times2^2+0\times2^1+1\times2^0=8+0+0+1=9$

です．このようにしてどのような2進数も10進数に変換することができます．

逆に10進数を2進数に変える方法について説明します．たとえば10進数の27を2進数に変えてみます．その方法は27を次々に2で割っていくのです．図2．2を見てください．

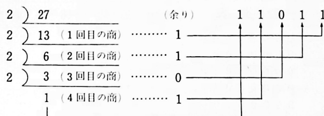


図2.2

このようにして計算して，商が1になったら，そこから矢印の方向に1，0を並べます．この結果は11011_2となります．これが2進数で表された10進数の27です．

ここでちょっとためしに先程覚えた2進数を10進数に変換する方法でこの11011_2を10進数にしてみます．

$11011_2=1\times2^4+1\times2^3+0\times2^2+1\times2^1+1\times2^0=16+8+0+2+1=27$

答えは27になりました．この例に従って自分で適当に問題を作って2進数←→10進数の変換をマスターして下さい．

2進数の桁の単位としてビット（Bit；Binary＝2進の Digit＝桁，の略号）という言葉を使います．たとえば101_2は3ビットの2進数，1101_2は4ビットの2進数という使い方をします．コンピュータにおいては，2進数をいくつかまとめて処理します．そのことを一度に4ビットの処理をするコンピュータとか8ビットを処理するコンピュータであると表現します．

コンピュータにはこのように一度に処理できるビット数としては，4ビット，8ビット，16ビット，32ビットなどといろいろありますが，我々がここで扱う TK-85 では一度に8ビットの処理をします．つまり，マイクロコンピュータにもいろいろあり TK-85というマイクロコンピュータでは8ビットを1単位として処理するものであるということです．なお8ビットのことを1バ

イト（Byte）ともいいます．そのため一度に８ビットの処理をするコンピュータのことをバイトマシンということもあります．

さて，２進数もビットが少ないうち，つまり桁数が少ないうちはよいのですが桁数が増てくると大変見にくくなってきます．10111101_2などと書いてあっても読みにくいし，書くのも大変です．コンピュータにとっては有利なことの多いこの２進数も人間にとっては扱いにくくなってきます．

そこで，10111101_2などを４ビットずつに区切って，それぞれの４ビットに対して16個の数字を対応させる16進数というものがよく使われます（４ビットで16通りの文字が表現できます）．つまり10111101_2を

[1011] [1101]₂

のように分け，表２．１の変換表に従って16進数にするのです．すなわち1011_2はＢ，1101_2はＤと表せます．したがって10111101_2は16進数で２桁のＢＤという数になりますね．

表２．１

２　進　数	10　進　数	16　進　数
0 0 0 0 0 0 0 0	0	0
0 0 0 0 0 0 0 1	1	1
0 0 0 0 0 0 1 0	2	2
0 0 0 0 0 0 1 1	3	3
0 0 0 0 0 1 0 0	4	4
0 0 0 0 0 1 0 1	5	5
0 0 0 0 0 1 1 0	6	6
0 0 0 0 0 1 1 1	7	7
0 0 0 0 1 0 0 0	8	8
0 0 0 0 1 0 0 1	9	9
0 0 0 0 1 0 1 0	10	A
0 0 0 0 1 0 1 1	11	B
0 0 0 0 1 1 0 0	12	C
0 0 0 0 1 1 0 1	13	D
0 0 0 0 1 1 1 0	14	E
0 0 0 0 1 1 1 1	15	F
0 0 0 1 0 0 0 0	16	10
⋮	⋮	⋮
0 0 0 1 1 1 1 1	31	1F
⋮	⋮	⋮

この16進数は，マイクロコンピュータを使っていくうえには不可欠なものですから，早く慣れるようにして下さい．

（16進数では０～９の数字のほかにＡＢＣＤＥＦといった文字が出ています．これは我々の使

っている数字は０～９までで，１桁で15まで表す数字がないため10以上の数字をアルファベットで代用したものです．したがってこのＡＢＣＤＥＦは数字です.）

また,以後16進数がでてきたら,1AH のように末尾にＨをつけて２進数や10進数と区別します.

次にマイクロコンピュータの扱える数の表現範囲の話をします．TK-85 では８ビット，つまり１バイトを１単位として処理することを書きました.

つまり一度に表現できる数の範囲は00000000_2から11111111_2まで，10進数でいうと０から255までの数なのです．ただし，これはマイクロコンピュータでは０から255までの10進数に相当するものしかあつかえないわけはなく，プログラムをうまく作ることにより，どんな数字でも扱うことができますから心配はいりません．さて，10進数にすると０～255までの数を表現できると書きましたが，負数はどのようにして表現されるのでしょうか？　幸いなことに数には正か負の状態しかありませんから，８ビットの２進数の最上位の桁（これを MSD：Most Significant Digit，といいます.）が０のとき正，１のとき負として，残りの７ビットで数を表すことにします.

こうすることによって，＋127から－128までを表現することができるようになります．ただし，負数の場合，MSD を１にしただけではだめで，残りの７ビットは補数で表現されている必要があります.

さて補数という考え方は初心者にとっては大変難解だと思います.

これは２進数を扱うコンピュータ（ディジタルコンピュータといいます）の宿命といってもよいでしょう.これはコンピュータの減算回路を加算回路で実現しようという考えによるためです.

わかりやすくするためまず10進数で説明します.

10進数１桁の数ａの10の補数とはａ＋ｂ＝10となるｂのことです．すなわち，１と９，２と８，３と７，４と６，５と５が互いに補数の関係にあります.

この補数の関係を用いて減算を加算で行なってみましょう．８－４の場合を考えてみます.

８－４を行う代りに８＋（４の補数；正確には４の10の補数）を考えます．そこで４の補数は６ですから８＋６＝14となります.ここで桁上げの１を無視すれば８－４と同じ結果になります.すなわち減算の場合，引く数の補数を加えて桁上げを無視すればよいことになります．しかし桁上げが起こらないこともあります.この場合は結果が負の数になるときです.たとえば４－８＝－４は，補数を用いて計算すると４＋２＝６となります．そしてこの６の補数４が求める数の絶対値です.

それでは２進数による補数の表し方について説明しましょう.

10進数の－８をディジタルコンピュータで表現するとき，まず８を２進数にします.

$8 = 00001000_2$（１バイト表現）

そしてすべてのビットを０なら１へ，１なら０へ変えます.

$00001000_2 \Rightarrow 11110111_2$

そしてその結果に１を加えます.

$$
\begin{array}{r}
11110111_2 \\
+ \qquad\quad 1_2 \\
\hline
11111000_2
\end{array}
$$

この11111000_2が補数の考え方による－8なのです．つまり，負数表現をしたい10進数の絶対値の数を2進数にして，すべてのビットを反転して1を加えることが2進数で補数表現をする手続きだと覚えておいて下さい．

では，この方法を使って先ほどの8－4＝4および4－8＝－4を2進数に直して計算してみましょう．

8－4の場合

8の2進数00001000_2を求めます．

4の2進数00000100_2を求め，さらにその補数11111100_2を求めます．

$$
\begin{array}{l}
\quad 00001000_2 \\
+11111100_2 \\
\hline
100000100_2 \longrightarrow 00000100_2
\end{array}
$$

100000100_2⟶00000100_2すなわち10進数の4
↑
桁上げを無視する

4－8の場合

4の2進数00000100_2を求めます．

8の2進数00001000_2を求め，さらにその補数11111000_2を求めます．

$$
\begin{array}{l}
\quad 00000100_2 \\
+11111000_2 \\
\hline
\quad 11111100_2
\end{array}
$$

11111100_2⟶00000100_2→10進数4の負の数－4
（↑ 補数を求める）
↑ 桁上りがないので負の数となります．

このようにして減算を加算として実行することができます．

3．8085 CPU の話

TK-85ではCPUに型番号が8085と呼ばれるCPUを使用しています．この8085というCPUは1974年アメリカのインテル社から発売された8080というCPUの改良発展型です．8080 CPUは比較的早い時期に作られたCPUのため多くのユーザーに使われました．そのため，世の中に最も広く使われているCPUとなっています．8085はこの8080をより進歩されたものですが，それはハードウェア（コンピュータ本体の回路）の改良であり，ソフトウェア（コンピュータの使用方法やプログラム）の点ではほとんど同じです．

そのため8080で作られたソフトウェアがほとんど変更なしに8085にも使用できます．人件費の高い現在では大変手間のかかるソフトウェアの開発にはぼう大なお金がかかりますから，いまま

でに開発されたソフトウェアがそのまま自由に使えるということはユーザーにとっても大きな利点となります．

それでは，これから8080の発展型である8085の中味をのぞいて見ましょう．

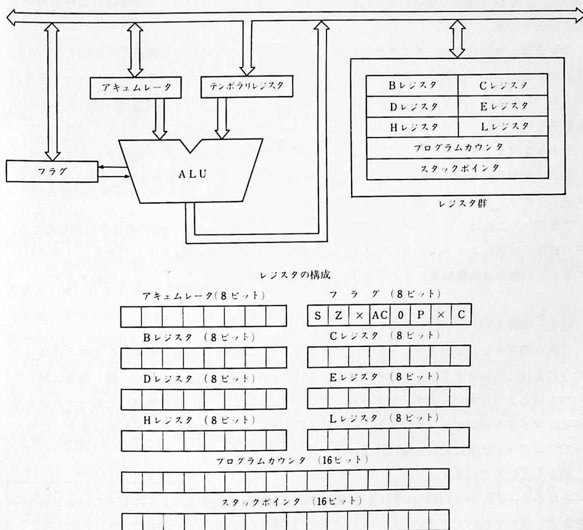


図2.3

8085 CPU は図2．3に示したようになっています．以下主な部分を順番に説明することにします．

レジスタ——マイクロコンピュータがいろいろな仕事をするとき，メモリのほかにデータを一時的に記憶をする部分と考えて下さい．

8085では8ビットのB，C，D，E，H，Lレジスタをもっています．なお，BとC，DとE，HとLをペアにして16ビットのレジスタとして使用することもできます．

アキュムレータ——Aレジスタとも呼ばれ，各種の算術や論理演算等を実際に行うときに用い

られるレジスタです．B，C，D，E，H，Lレジスタとは多少異なった特徴をもっているため アキュムレータという名前で呼ばれます．

ALU――CPU の中で算術演算と論理演算を行うユニット(ALU：Arithmetic and Logical Unit の略）で，その入力の一方にアキュムレータ，他方にテンポラリレジスタと呼ばれる作業用レジスタがあります．

プログラムカウンタ――マイクロコンピュータはプログラムに従って動いているわけですから，現在何番地のメモリに書いてあるプログラムを実行しているのかをマイクロコンピュータ自身が知らなければこまります．つまり次に実行すべきプログラムがメモリの何番地に書いてあるのかを知るもの，それがプログラムカウンタです．プログラムカウンタは16ビットのレジスタです．

スタックポインタ――これも16ビットのレジスタです．これはのちほど説明されるサブルーチンなどの利用法において，各レジスタの内容を保存しておくメモリの番地を示しておくものです．スタックポインタの詳しい使い方は後で述べます．

フラグ――これは8ビットのレジスタです．フラグレジスタはある特定の命令を実行したとき，その結果の状態によって1か0になるレジスタです．たとえば演算命令をしたときにフラグレジスタのZの部分は演算結果が0のとき1になり，そうでなければ0というように動きます．

以上，8085 CPU の中味の概略を述べてきました．

これらのアキュムレータやレジスタはプログラムにより示される命令を実行してゆくに従って次々に変化していきます．しかしその変化は決してあやふやな変化ではなく，同じ命令に対していつも同じように変化します．ですから，プログラムの中に，少しでも間違った部分があるとその後はマイクロコンピュータは（我々から見て）メチャクチャな動きをしてしまいます．したがってプログラムは常に正確でなければいけません．

話をもとにもどします．

とにかく，TK-85 に使われている 8085 CPU の中味は以上述べたようになっているということをよく覚えておいて下さい．

次にメモリ空間ということについてお話ししましょう．先程プログラムカウンタは16ビットのレジスタであると書きました．これはプログラムカウンタは16桁の2進数を表すことができるということなのです．つまり，0000000000000000_2から1111111111111111_2までの2進数を表現できるということです．10進数にしますと0から65535まで表現できることになり，メモリの番地を10進数で0～65535番地まで扱うことができます．8085は一度に8ビットの2進数を処理できます．つまり一度に1バイトを処理できますから，最大では65536バイトのメモリを操れることになります．

ただし標準のTK-85はROM 2048バイトRAM 1024バイトのメモリ装備ですが，マイクロコンピュータの勉強をするにはこれで十分です．マイクロコンピュータの知識を身につけていくうちに，8085 CPU の最大メモリ空間である65536バイトまで TK-85 を拡張することも可能になってきます．

4．8085 CPU の命令について

メモリにはプログラムが書かれている話はすでにしました．しかし，メモリのある一部を見たときに，それが必ずしも CPU への命令が書かれているとはかぎりません．それはデータ（情報）かもしれません．つまりコンピュータのメモリに記憶されている内容は CPU に対する命令だけではないのです．

たとえば，コンピュータそのものはアイウエオの五十音や，ＡＢＣなどの文字をもっているわけではないのに，実際のコンピュータではいかにも扱われているように見えます．TK-85 の CPU 8085 では，一度に処理できる１バイト単位であらかじめ決められた約束事に従って文字に対応する２進数を決めておき（たとえば文字Ａは２進数の01000001_2というように），この２進数をメモリに必要に応じて記憶させておきます．そしてデータとしてＡに対応する２進数がでてきたとき，これはＡという文字だと判断していくのです．

このようにメモリの中にあるのは命令だけではないということを理解したうえで，プログラムの中における命令はどのようなものがあるかを２，３の例とともにお目にかけることにします．なお，命令の解説では16進数を多用します．マイクロコンピュータではプログラムでも16進数は大変よく使われますから16進数に慣れるようにして下さい．

まず，アキュムレータに 4CH というデータを入れる命令について考えましょう．図２．４を見ながら以下の説明を読んで下さい．

TK-85 マイクロコンピュータではこの命令のことを MVI A, 4CHと表現（ニーモニックコード）され，メモリの0001H番地３Ｅ（0011110）として入っています．つまり，8085 CPU はプログラムを実行している最中にプログラムカウンタが示した番地にこの命令があったら，そのメモリの次の番地にあるデータ 4CH をアキュムレータに入れるということです．

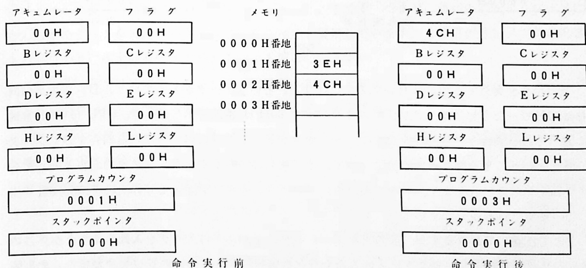


図2.4

おわかりでしょうか？　CPUはプログラムカウンタにより，次に実行する命令の入っているメモリ番地0001Hをおしえてもらい，そこに書いてある3EHを読み，その命令は，次のメモリ番地に書いてある内容をアキュムレータに入れることだと知り，それを実行します．そして，その実行が終わるとプログラムカウンタは次に実行する命令がはいっている0003Hを示しており，CPUは次の命令の実行のために0003Hの内容を読みにいこうとします．

言葉でいうと大変長くなりますが図と説明をよく読めばおわかりいただけると思います．

ここで一言ふれておきますが，この動作は人間の目から見ると一瞬のうちに終わります．このアキュムレータに4CHという内容を入れる命令を実行するのにかかる時間はわずか約2.8×10^{-6}秒，つまり1秒間にこの命令ばかりを繰り返すなら35万回もできるほど高速なのです．

もう一つの例を示します．

Bレジスタに入っているデータをDレジスタに移すことを考えてみます．この命令はメモリの中に16進数のかたちで50Hとして入っています．図2．5を見ながら以下を読んで下さい．

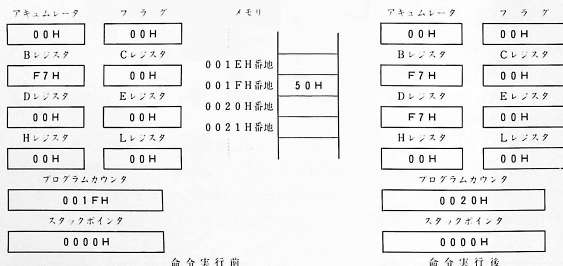


図2.5

この命令を実行する前にはBレジスタにはF7Hというデータが入っていて，Dレジスタの内容は00Hだったとします．プログラムカウンタが001FH番地を示したとき，CPUは001F番地に書いてある50Hという内容を読んで，その命令がBレジスタにはいっている内容をDレジスタに移すということを知りそれを実行します．その結果は当然ながらBレジスタの内容と同じデータがDレジスタに入ります．命令の実行後はプログラムカウンタはメモリの次の番地0020Hを指しています．おわかりですか？

ここで注意しておきますが，前のアキュムレータになんらかのデータを入れる命令とか，このBレジスタの内容をDレジスタに入れる命令などを実行しても，ほかのレジスタ類はなんの影響も与えないということです．この図2．4，図2．5の例では，命令の実行において直接関係のな

いレジスタなどはすべて00Hにしてありますが，レジスタの内容が00H以外でも同様なことがいえます．つまり命令に関係ないレジスタは変化しません．

さらにもうひとつの命令も見てみましょう．

アキュムレータにメモリの 1F3BH 番地に入っているデータを入れることを考えてみます．

図２．６を見てもらいます．プログラムカウンタは 11FCH 番地を指しています．CPU は 11FCH 番地の内容32Hを読み，その次の番地の内容を下位アドレス，そのまた次の番地の内容を上位アドレスとして，この上位，下位アドレスで示されるメモリ番地に書いてある内容をアキュムレータに入れる命令だということを知り，それを実行します．

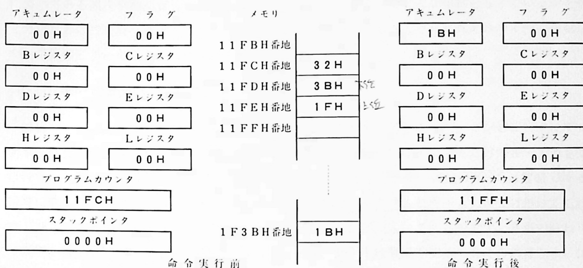


図2.6

以上の三つの命令についてながめてみましたが，こんなことに気付きませんでしたか？　命令はいくつかの長さの種類があるのではないかということです．つまり，最初のアキュムレータになんらかのデータを入れる命令はメモリを２バイト使用しています．２番目のBレジスタの内容をDレジスタに移す命令はメモリを１バイト使っています．

このように命令にはメモリを１バイトだけ使う１バイト命令，２バイト使う２バイト命令，３バイト使う３バイト命令の３種があることをここでは覚えておいてください．

8085 CPU の命令はこのほかにもいろいろあります．しかし，他の命令もいま説明した三つの命令のようにそれほどむずかしくはありません．

読者の中には，「このような命令がいくつかあることはわかったけれども，どうしてこんな命令をすることで，コンピュータはあんなむずかしい仕事をすることができるのだろう」と考えておられる人もいるかもしれません．しかし，コンピュータは命令をうまく並べること，つまりプログラミングをすれば，単純な個々の命令からは想像もつかないむずかしい仕事もできます．その方法については次章以降で順々にお話ししていくことにしましょう．

3章　TK-85の操作方法

1. はじめに

1章において TK-85 を自分の手で動かしてみました．しかし，その動かしかたは，ただ本書に書いてある操作手順に盲目的に従っていくだけで押したキーがどんな役割をもっているかを知っていたわけではありませんでした．この章では TK-85 のもっているそれぞれのキーの役割について説明していくことにします．

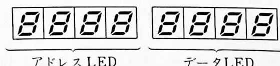
2. キーの説明をする前に

前にコンピュータはプログラムがなければ動かないことを説明しました．コンピュータはプログラムがなければなにもできないのです．では TK-85 は電源を ON にするだけでキー入力を受け付けてくれたり，LED にデータを表示してくれるのでしょうか？　実は TK-85 は電源 ON と同時に動くプログラムをもっているからなのです．このプログラムにキー入力を受け付けてくれるプログラムや LED にデータを表示させるプログラムなどが入っているのです．0 章の終わりで説明した ROM にこのプログラムが入っています．ROM は電源を切ってもその内容をうしなうことはありません．ですからこのように電源が入ったら TK-85 がすぐに使えるようにするプログラムなどを入れるのに向いています．話をもどします．

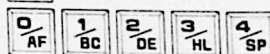
このように電源 ON と同時に TK-85 をあるレベルまで操作しやすくしてくれるプログラム，このプログラムをモニタプログラムといいます．

そして，TK-85 がモニタプログラムを実行しているとき，TK-85 はモニタの管理下にあるといいます．

次に LED について述べます．以下の説明では LED の左側の 4 桁をアドレス LED，右側の 4 桁をデータ LED と呼びます（下図参照）



3. データ入力

TK-85 は電源を入れて RESET ボタンを押した直後，あるいは動作の途中に RESET ボタンを押した直後は必ずデータ入力が可能な状態になっています．さて，データ入力とは 

[5/BRP][6/BRD][7][8][9][A/SAVE][B/LOAD][C/TM][D/MOV][E/OUT][F/IN]のそれぞれのキーを押して16進数のデータを TK-85 に入力することです．ではまずその操作をしてみます．

TK-85 の電源を入れ，[RESET]ボタンを押します．

以下を見て下さい．

キー入力	アドレス LED	データ LED
RESET	0000	0000
1/BC	0000	0001
2/DE	0000	0012
3/HL	0000	0123
4/SP	0000	1234
5/BRP	0000	2345
6/BRD	0000	3456

このように入力されたデータはデータ LED の右端から表示されていき，データ入力があるたびに左に桁上がりしていきます．そしてデータ LED がいっぱいになるとデータ LED の左端のデータはなくなりあとから入力した 4 桁のデータのみが有効になります．上図で見れば，[5/BRP]のキーを入力したときには[2345]と表示されていますが，このときには[1/BC]の入力は無効となっていて，[2/DE][3/HL][4/SP][5/BRP]の入力だけが有効になっています．

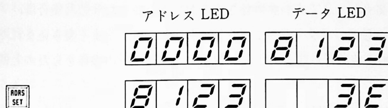
4　NORMAL FUNCTION

TK-85 の操作は大きく分けて「NORMAL FUNCTION」「MODE FUNCTION」「REGISTER FUNCTION」の三つに分かれます．では最初に NORMAL FUNCTION から説明します．

4．1　ADDRESS SET AND MEMORY READ ([ADRS SET])

このキーはデータ LED に表示されているデータをそのままアドレス LED に移し，そのデータを番地とするメモリの内容をデータ LED に表示します．

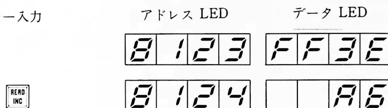
次の図を見てください．



この場合，データ LED には8123Hというデータが入力されています．そこで[ADRS SET]キーを押すと，そのデータはアドレス LED にそっくり移り，データLEDには8123H番地のメモリの内容である 3EH（これがなにになるかはあなたの TK-85 による．ここでは 3EH としただけです.）が表示されます．

4.2 ADDRESS INCREMENT AND MEMORY READ（[READ INC]）

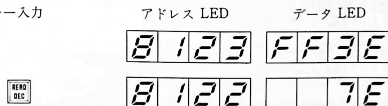
アドレス LEDに表示されているデータを＋1して，その番地のメモリの内容をデータ LED に表示します．



上図ではアドレス LED に8123Hが表示されていたところへ[READ INC]を押したので，8123Hに＋1した8124H番地がアドレス LEDに表示され，データ LED にはその番地の内容 A6H（この内容も[ADRS SET]キーの説明と同じ）を表示します．

4.3 ADDRESS DECREMENT AND MEMORY READ（[READ DEC]）

このキーは4.2項の[READ INC]キーとは逆の働きをします．つまりアドレス LEDに表示されているデータを－1して，その番地のメモリの内容を読み出すのです．その結果，アドレス LEDには－1されたデータが，データ LEDにはその番地の内容が表示されることになります．



上図ではアドレス LED には8123Hが表示されています．[READ DEC]キーを押すと8123Hから－1をしたデータをアドレス LED に，その番地のメモリの内容 7EH をそれぞれ表示します．

4.4 MEMORY WRITE AND ADDRESS INCREMENT（[WR ENT]）

アドレス LED の表示内容により指定されている番地のメモリに，データ LED に表示されている下位2桁のデータを書き込みます．もし，指定されたメモリが書き込みのできない ROMで

あったり，RAM が異常でメモリにうまくデータの書き込みができなかったりした場合は，アドレス LEDに［ ］［E］［r］［r］と表示して，エラーがおこったことを示します．うまく書き込まれたときは続いて（このときは一瞬のうちに行われます．）先に説明した［READ INC］キーの操作をしたのと同じことが行われます．

キー入力	アドレス LED	データ LED
	8123	FF3E
［WR ENT］	8124	A6

上図の場合，アドレス LED に表示されている8123H番地のメモリに，データ LED の下位2桁に表示されている 3EH という内容が書き込まれ，同時に8123Hに＋1した8124H番地がアドレス LED に，そのメモリ番地の内容がデータ LED に表示されます．

- ここでいままで覚えたキー操作の応用をお目にかけます．メモリの8123H番地にデータ 1FH を，8124H番地にデータ 2DH を書き込みたいとします．その操作手順はどのようになるでしょう．操作手順は以下のとおりです．

キー入力	アドレス LED	データ LED
［RESET］	0000	8123
［8］［1 BC］［2 DE］［3 HL］		
［ADRS SET］	8123	UN UN
［1 BC］［F IN］	8123	UN UN 1F
［WR ENT］	8124	UN UN
［2 DE］［D MOV］	8124	UN UN 2d
［WR ENT］	8125	UN UN

本当にメモリに書き込まれたかどうかをたしかめるには［READ DEC］キーを使えばよいですね．上記のキー操作に続けて［READ DEC］キーを押せば，押されるごとにアドレス LED に表示されているメモリ番地がひとつずつ－1されていき，データ LED にその番地の内容が表示されますから，この場合ですと，8124H番地に 2DH が，8123H番地に 1FH が書き込まれているかがすぐわかります．

あるいは他の方法として，8123H番地の内容を［ADRS SET］キーを使って LED に表示させ，［READ INC］キーを使うことも可能です．これは読者のみなさんが自分でやってみて下さい．

これまでの説明で，もうメモリにデータを書き込んだり，読み出したり，あるいは修正することができるようになりました．今まで説明してきませんでしたが，TK-85は標準装備でモニタプログラムにより自由に読み書きのできるメモリは8000H番地から8390H番地までです．読者のみなさんはこのメモリの範囲内でメモリにデータを書き込んだり読み出すことの練習をして下さい．

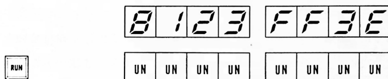
4．5 RUN（RUN）

この章の最初でモニタプログラムの話をしました．つまり我々が TK-85 のキーを押しているとき，TK-85 の 8085 CPU はこのモニタプログラムを実行しているわけです．2．2．4項で書いたとおりに我々が TK-85 で自由にあつかえるメモリは8000H番地から8390H番地までです．

ではここに我々がプログラムを書いた場合，どのようにしてこのモニタプログラムから我々のプログラムへ 8085 CPU の実行を移すのでしょうか？，そのときのキーが RUN キーです．

アドレス LED に表示されているデータを番地とするメモリへ 8085 CPU の実行を移す，つまり，我々のプログラムを実行させるために我々のプログラムの書いてあるメモリ番地へ 8085 CPU を飛ばすのです．

キー入力　　　アドレス LED　　　データ LED



上図においては 8085 CPU は8123H番地から書いてあるプログラムを実行することになります．

4．6 RESET

このキーを押すと TK-85 はどんな仕事をしていてもモニタプログラムの管理下にもどります．

たとえば我々の書いたプログラムが少し間違っていたため，TK-85がおかしな動作をしてしまったときなど，この RESET キーを押せばモニタの管理下にもどります．それからそのプログラムの間違った所を直したりすることができます．

TK-85 がおかしな動きをしたとき RESET を押すと TK-85 の頭が正常にもどると思っておいて下さい．

4．7 MONITOR（MON）と CONTINUE（CONT）

4．5項で説明した RUN キーを使って我々のプログラムを TK-85 が実行しているとき，MON キーを押すと，8085 CPU のすべてのレジスタの内容を，ある所に保管しておきます．つまりアキュムレータ（Aレジスタともいう），フラグ，B，C，D，E，H，Lレジスタおよびプログラムカウンタ，スタックポインタの内容を保管しておくのです．そしてアドレス LED にはそのとき実行しようとしていたメモリ番地を表示し，データ LED には上位2桁にアキュムレータの8ビットのデータを16進数にした数字を，下位2桁にフラグの8ビットの内容を16進数にした内容を表示してモニタの管理下にもどります．

[CONT]を押したときには逆に保管してあったレジスタ等のデータをすべて8085 CPUにもどし，先程実行していたプログラムを再開します．(ただし，プログラムの内容を変化させなかった場合は正常に動くが，それ以外の場合には保障できません．)

5. MODE FUNCTION ([MODE])

[MODE]キーを押すとアドレスLEDに四つの点が表示され，MODE FUNCTIONに入ったことを知らせます．MODE FUNCTIONに入ると以下六つのキー[A/SAVE][B/LOAD][C/TM][D/MOV][E/OUT][F/IN]のどれかを選択しなければなりません．

キー入力	アドレスLED	データLED
	8123	FF3E
[MODE]	. . . .	

5.1 IN ([MODE][F/IN])

ここで説明しますINと次の5．2項のOUTはTK-85のハードウェア（おもに回路などの意味）の知識が必要となりますので，まだTK-85のハードウェアについて詳しく知っていないかたはこの本の11章を先に読んでからここを読んで下さい．ここではそれらの知識があるものとして書きます．

INモードではI/Oポート（INPUT & OUTPUT）からデータを入力するときに用います．I/Oポートとは入出力機器のことだと思われて結構です．

[MODE][F/IN]と続けて押すとINモードになり，アドレスLEDにそれを表示します．

その後IN PORT ADDRESS（つまり入力する機種の選択です．）を16進数2桁で入力し[WR/ENT]キーを押すとIN PORT ADDRESSからデータを入力しデータLED下位2桁に表示します．

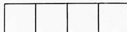
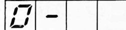
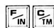
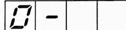
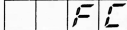
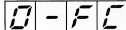
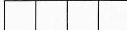
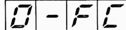
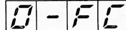
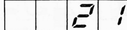
キー入力	アドレスLED	データLED
[MODE]	. . . .	
[F/IN]	1-	
[F/IN][1/BC]	1-	F8
[WR/ENT]	1-F8	dF
[WR/ENT]	1-F8	dF

上記ではIN PORT ADDRESSにF1Hを選びました．するとそこから入力されたデータ74Hがデータ LED 下位2桁に表示されました．続けて[WR ENT]を押すと新しい入力データの76Hが表示されました．(この入力データは別に意味はなく，みなさんのTK-85ではどのように表示されるかわかりません．)

5．2 OUT ([MODE][E OUT])

OUT モードはI/Oポートへデータを出力するときに使います．[MODE][E OUT]キーを押すとOUTモードになりアドレスLEDにそれを表示します．

続けてOUT PORT ADDRESSを16進数2桁で入力し，[WR ENT]を押し，さらに出力するデータを16進数2桁で入力して[WR ENT]を押すとそれが出力されます．ひき続き同じOUT PORT ADDRESSにデータ出力をする場合には[WR ENT]を押すか，新しいOUTデータを入力してから[WR ENT]を押します．

キー入力	アドレス LED	データ LED
		
		
		
		
		
		

上図ではOUT PORT ADDRESSにFCHを選び，OUTデータに21Hを選んで出力しています．

5．3 MOVE MOEMORY ([MODE][D MOV])

このモードは，メモリに書いてある内容をそっくりそのまま違う番地のメモリに移す働きをするモードです．たとえば8000H番地から8150H番地までに書いてある内容を，8100H番地から8250H番地までのメモリに移すなどです．

[MODE][D MOV]と押すとMOVEモードになります．MOVEに対応するドット(点)だけが点灯してキー入力状態になります．まずメモリの中の移したい部分の先頭番地を16進数4桁で入力し[WR ENT]キーを押します．続いて最終番地を16進数4桁で入力し[WR ENT]キーを押します．このとき，先頭番地が最終番地より大きいときはエラーメッセージが出され，モニタプログラムの管理下にもどり，MOVEモードは解除されます．

次に移すメモリの先頭番地を16進数 4 桁で入力して [WR/ENT] を押すとメモリデータの転送が行われます．なお，この動作全体をメモリのブロック転送といいます．

さて転送が終了すると転送されたメモリの最終番地が表示され，MOVE モードのアドレス LED のドット表示が消えブロック転送が終わったことを示します．

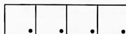
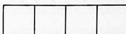
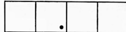
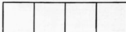
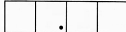
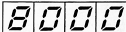
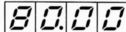
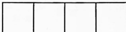
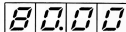
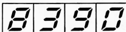
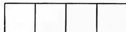
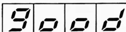
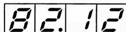
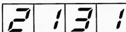
キー入力	アドレス LED	データ LED
[MODE]	.　.　.　.	
[D/MOV]	.	
[8] [0/AF] [0/AF] [0/AF]	.	8000
[WR/ENT]	8.000	
[8] [0/AF] [5/ERP] [0/AF]	8.000	8050
[WR/ENT]	.	
[8] [1/BC] [0/AF] [0/AF]	.	8100
[WR/ENT]	8.100	
(転送終了)	8100	8150

上図では8000H番地から8050H番地までのデータを8100H番地から8150H番地まで転送しています．

5.4　TEST MEMORY （[MODE] [C/TM]）

TEST MEMORY モードは TK-85 の RAM が正常に働いているかのチェックに使われます．将来，TK-85 の RAM を増設したときなどに使用されることもあります．

[MODE] [C/TM] と押すと TEST MEMORY モードに入ります．続いてテストするメモリの先頭番地を16進数 4 桁で入力して [WR/ENT] を押し，さらに最終アドレスを16進 4 桁で入力して [WR/ENT] を押すとテストメモリが始まります．テストが正常に終了するとデータ LED に good と表示，そうでないときにはアドレス LED にその異常のあったメモリ番地，データ LED の上位 2 桁に書き込んだデータ，下位 2 桁に読み出されたデータが表示されます．ひき続きテストメモリをするときには [WR/ENT] キーを押せば可能です．なお，このテストメモリを実行させるとそのRAMの内容はデタラメになりますので注意して下さい．

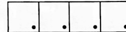
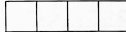
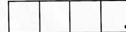
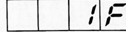
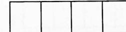
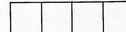
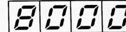
キー入力	アドレス LED	データ LED	
			
			
   			
			
   			
			テストメモリ終了
			異常のとき

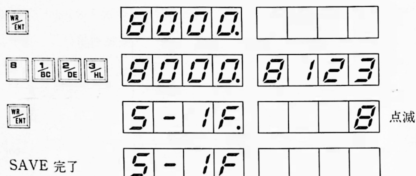
5.5 CASSETTE SAVE（[MODE][A SAVE]）

これはメモリの内容をカセットテープに録音するモードです．

[MODE][A SAVE]と押すと SAVE モードになります．

まず，録音するデータの名前にあたるファイル名を入力します．16進2桁で入力して[WR ENT]を押します．これは00HからFFHまでの256通りがあります．次にSAVEするデータの先頭アドレスを16進4桁で入力して[WR ENT]を押し，続いて最終アドレスを16進4桁で入力します．テープレコーダとTK-85がしっかりつながっているかを確認したらテープレコーダを録音状態にして[WR ENT]を押します．データ LED の最下位が点滅してSAVEが終了するとSAVEモードが解除されます．

キー入力	アドレス LED	データ LED
		
		
 		
		
   		



上図では8000H番地から8123H番地までのデータをSAVEしています.

5.6 CASSETTE LOAD ([MODE] [B/LOAD])

[MODE] [B/LOAD] と押すとLOADモードとなります.

LOADファイルのナンバーを16進2桁で入力します. [WR/ENT] を押してテープレコーダを再生させます. LOADファイルナンバーと異なるファイルナンバーを見つけるとデータLED下位2桁に表示します. LOADしたいファイルを見つけるとアドレスLEDにその表示をしてロードを開始します. ロードが正常に終わるとロードしたデータの先頭番地と最終番地をそれぞれアドレスLEDとデータLEDに表示します. ロードがうまくいかなかったときはアドレスLEDにエラーメッセージを出します.

キー入力	アドレスLED	データLED	
[MODE]	. . . .		
[B/LOAD]	. .		
[1/BC] [F/IN]	.	1F	
[WR/ENT] (検索中)	S - 1.F	02	(異なるファイルナンバー)
(ファイル発見)	F - 1F	8	(点滅)
(終了)	8000	8123	
(異常あり)	E - 1F		

⑥. REGISTER FUNCTION ([REC])

[REC]を押すとデータ LED の最上位のドットが点灯し，REGISTER FUNCTION に入ったことを知らせます．このモードになると次の七つのキー[0/AF][1/BC][2/DE][3/HL][4/SP][5/BRP][6/BRD]の選択が必要になります．

キー入力	アドレス LED	データ LED
		.

[REC]キー入力後，上記のいずれかのキーを押すとアドレス LED に表示レジスタ名，データ LED にその内容を表示します．表示レジスタの内容を変更したいときには16進データを入力して[WR/ENT]キーを押します．16進データを入力せずに[WR/ENT]キーを押すと次のレジスタ表示に移ります．

後で述べる BREAK 動作における BREAK POINT および BREAK DEPTH の設定はこの FUNCTION で行います．

キー入力	アドレス LED	データ LED
		.
	bC--	0000
	bC--	8033
	dE--	0000

AF
BC
DE
HL
SP
BR.P
BR.D

上記においては TK-85 の 8085 CPU の BC レジスタの内容をデータ LED に 4 桁で表示します．Bレジスタの内容は上位 2 桁に，下位 2 桁にCレジスタの内容を表示します．上記例では B レジスタに80Hというデータを，Cレジスタには33Hというデータに修正したときの様子です．他のレジスタでも同様になります．読者のみなさんで試して下さい．なお，この FUNCTION で表示できるのは「アキュムレータ&フラグ」「B&Cレジスタ」「D&Eレジスタ」「H&Lレジスタ」「スタックポインタ」「BREAK POINT」「BREAK DEPTH」の七つです．

⑦. ステップ動作

TK-85 のモニタプログラムでは，我々の作ったプログラムを1命令ずつ実行させることができるようになっています．これはプログラムが正しく書かれているかのチェック（デバッグ＝虫と

り，ともいう）などに便利です．

これらの動作のことをステップ動作をさせるといいます．

ステップ動作を行うには，まず RESET キーあるいは MON キーを押して BREAK DEPTH を 0 にする必要があります．BREAK DEPTH については次の 8 節ブレイク動作で説明します．

なお，この BREAK DEPTH を 0 にするには 6 節で述べた REGISTER FUNCTION を使う方法もあります．BREAK DEPTH を 0 にしたらステップスイッチを STEP 側に倒します．この状態でアドレス LED にプログラムの先頭番地をセットして RUN キーを押しますとそのプログラムを 1 命令実行してモニタプログラムにもどります．

このとき，CPU のレジスタの内容はすべて保管されています．そしてアドレス LED にはプログラムカウンタの内容，データ LED の上位 2 桁にはアキュムレータの内容，下位 2 桁にはフラグレジスタの内容が表示されます．

この後 CONT キーを押しますと保管されていたレジスタの内容をすべて復帰させ，次の 1 命令を実行して再びモニタプログラムにもどってきます．

さらに CONT キーを押すとこのステップ動作を引き続き実行できます．

8.　ブレイク動作

TK-85 のモニタプログラムでは BREAK POINT というものを設定することができます．

つまりこの BREAK POINT にセットしたメモリ番地においてプログラムの実行を BREAK つまり中断することができるようになっているのです．さらに BREAK POINT のほかに BREAK DEPTH というものももっています．

これによって BREAK POINT にセットしたメモリ番地を何回通過したら BREAK するかもセットできるのです．

まず BREAK POINT と BREAK DEPTH を REGISTER FUNCTION を用いて設定します．この場合 BREAK DEPTH は必ず 1 以上にして下さい．

たとえば BREAK POINT ですぐにプログラムをストップしたい場合には，そのメモリ番地を BREAK POINT にセット，そして BREAK DEPTH を 1 にすればよいのです．

また BREAK POINT を何回か実行したあとで止めたいなら，その実行させたい回数に＋1 した16進数を BREAK DEPTH に設定すればよいのです．なお，BREAK POINT にセットするメモリ番地は必ず各命令のオペレーションコードのあるメモリ番地でないといけないことに注意して下さい（このことについては次章を読んで下さい）．

以下に BREAK POINT と BREAK DEPTH の応用例を書いておきます．

図 3．1 ではオペレーションコードのことをオペコードと書いてあります．もしわからない場合は 4 章を読んでからここを読み直して下さい．

●ＢＲＥＡＫ動作の正しい使い方例

（1）指定ＢＲＥＡＫ ＰＯＩＮＴですぐ止めたい場合

BREAK POINT =8008H
BREAK DEPTH = 1

実行回数			
1	8000	オペコード1	B_2
1	8002	〃 2	B_3 B_2
1	8005	〃 3	B_3 B_2
0	8008	〃 4	
0	8009	JMP	8000H

（2）複数回実行後ＢＲＥＡＫ ＰＯＩＮＴで止めたい場合

BREAK POINT =8008H
BREAK DEPTH = 3

実行回数			
3	8000	オペコード1	B_2
3	8002	〃 2	B_3 B_2
3	8005	〃 3	B_3 B_2
2	8008	〃 4	
2	8009	JMP	8000H

●ＢＲＥＡＫ動作の悪い使い方例

（1）プログラムの1命令ですぐに止める場合

BREAK POINT =8000H
BREAK DEPTH = 1

実行回数			
無限ループ	8000	オペコード1	B_2
	8002	〃 2	B_3 B_2
	8005	〃 3	B_2 B_3
	8008	〃 4	
	8009	JMP	8002H

（2）オペコード以外が格納されているＡＤＤＲＥＳＳを指定した場合

BREAK POINT =8006H
BREAK DEPTH = 1

実行回数			
無限ループ	8000	オペコード1	B_2
	8002	〃 2	B_3 B_2
	8005	〃 3	B_3 B_2
	8008	〃 4	
	8009	JMP	8000H

（3）ＢＲＥＡＫ ＤＥＰＴＨが0の場合

BREAK POINT =8008H
BREAK DEPTH = 0

実行回数			
1	8000	オペコード1	B_2
0	8002	〃 2	B_3 B_2
0	8005	〃 3	B_3 B_2
0	8008	〃 4	
0	8009	JMP	8000H

図3.1

4章　プログラムの基礎と作り方

1．はじめに

1章でTK-85を動かしてみました．TK-85を動かしたプログラムは矢印プログラムというものでした．この章ではあの矢印プログラムはどのような手順で作られたか，どのような命令が使ってあるかを通じて8085 CPUの命令体系や，プログラムの作り方などについて考えてみることにします．

2．プログラムとは何か

今までに何度か述べてきましたが，プログラムとはコンピュータの基礎的な命令をある法則に従って順序正しく並べてあるものです．そして，それだけではなく，コンピュータがこの命令の順序通りに実行していくと，与えられた仕事が確実にできるようになっていなければなりません．コンピュータという機械を働かせるには上記のプログラムというもの，つまり仕事の手順が書いてあることが必要です．

我々がプラモデルを作るときのことを考えてみましょう．プラモデルを作るにあたってまずそれの組み立て説明書を見ます．その説明書にはたとえば次のように書いてあるでしょう．

1．部品に接着剤をつけなさい．

2．それを指定された所へとり付けなさい．

3．その上にシールを貼りなさい．

我々はこれを読んで，そのとおりにプラモデルを組み立てます．そのとき，組み立て説明書が間違っていたら，その間違い通りに組み立ててしまいます．

上の例を，我々がコンピュータ，組み立て説明書をプログラムに置き換えてみると非常に良く似ています．組み立て説明書が間違っていたら（プログラムが間違っていたら）とんでもないことになってしまいます．組み立て説明書がなかったら（つまりプログラムがなかったら）プラモデルの組み立ては不可能でしょう．

さて，コンピュータのできることは前にも書いたとおり基本的なものばかりです．ですからコンピュータを使用してある仕事をさせるためのプログラムを作るとき，その仕事を複雑なものから簡単なものに，そして最後にはコンピュータのできる命令のレベルまで分解していきます．そこまで仕事を分解できればコンピュータの命令をある法則に従って並べ，プログラムを作ることができます．

また，プログラムを作るときに，使用するコンピュータがどんな命令を持っているかを知らなくてはある仕事を分解していくときにも目安がありません．ですから，TK-85でプログラムを作るときには，TK-85の8085 CPUの命令について知っておく必要があります．またその命令をいかに並べるかについても知っておかなければなりません．

1章において矢印プログラムの中味そのものをお見せいたしましたが，あの矢印プログラムも全体の仕事を細分化してTK-85にできる命令まで分解して，それを並べたものなのです．

このようにプログラムを作るのは結構大変な作業です．しかし，このプログラムを作ることは今のところコンピュータにはできません．プログラムを作るには創造力や構成力が必要なのですが，コンピュータはそれらを持っていないか，あるいは非常に低いものしか持っていないのです．

ですから，今のところコンピュータのプログラムは人間が作る必要があります．ただし，そのプログラムを作る作業も人間に楽なようにするためいろいろなアイデアがだされ，そして実用化されてきました．

たとえば，人間にとってわかりやすい文法に従ってプログラムを書くと，それをコンピュータにわかるようにしてくれるプログラムなどがあります．このプログラムのことを一般にはプログラム言語といっています．

このプログラム言語にはいろいろな種類があります．Fortran（フォートラン）やCobol（コボル）などが有名です．マイクロコンピュータの世界ではBASIC（ベーシック）というプログラム言語が盛んに使われています．TK-85で使用できるプログラム言語はありません．直接，TK-85にわかるようにプログラムを書きます．TK-85に直接わかる言葉という意味で，このことをマシン語とか機械語といいます．つまりTK-85では機械語を使用してプログラムを書きます．

機械語は人間にとっては不便なことの多い言語ですが，マイクロコンピュータを学ぶには最適な言語ですし，将来マイクロコンピュータを応用するときなどにも大変役立ちますから，早く慣れることが必要です．もちろん，TK-85も拡張すれば機械語より便利なプログラム言語を使用することもできますから，あとでプログラム言語を使う楽しみを得ることもできます．

以上，プログラムについての概略をはなしてきましたが，いかがでしたか．プログラムについての概念が得られたでしょうか．

では次の節より8085 CPUのもっている命令についての勉強をしていくことにします．

3．プログラムの基礎

3.1 フローチャート

プログラムの勉強をしていくとわかりますが，プログラムのアルゴリズム（考え方）を考えていく作業は簡単ではありません．それはプログラムのアルゴリズムを考えるときに必要とする要素が多いため，いきなりプログラムを作りはじめると，必要な部分が欠けてしまったり，不要な部分があったりしてしまいます．そこで，プログラムを作る前にそのプログラムで実現させる仕

事に必要な事項や順序などをはっきりさせ，図でもって整理しておく必要があります．そのためにフローチャート（流れ図）が使われます．

フローチャートとは簡単にいうと仕事のやる順序を記号や矢印などで表していくものです．図4．1を見て下さい．

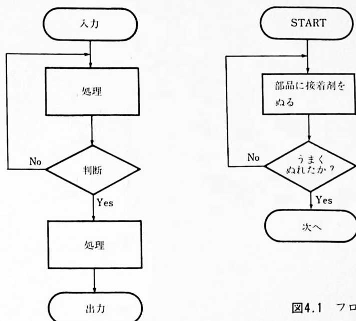


図4.1　フローチャート

左側が概念図，右側がプラモデルを作るときの様子をフローチャートにしてみました．

このような短い例ではフローチャートのありがたみはわかりにくいかもしれませんが，複雑な仕事をコンピュータにやらせるときなどフローチャートが書いてあるかどうかで，プログラム製作の能率は大きく異なります．

さらにフローチャートがあるとあとでプログラムを見直すときに大変便利です．同じようなことですが，他人の作ったプログラムを見るときにも便利です．フローチャートでプログラムの概略を見ながら，他人の作ったプログラムを解読するのもむずかしくありません．

このようにフローチャートはプログラムを作っていく上で非常に重要ですが，その使いかたは別にむずかしくありません．以下の文ではよくフローチャートがでてきます．フローチャートというものに慣れるようにして下さい．

3．2　コーディングシート

プログラムを実際に作るとき，普通はまず紙と鉛筆を用いてそれにプログラムを書くことになります．その書き方にはいくつかのルールがあります．ルールといっても厳密なものではなく，いろいろな種類がありますが，原則的なものだけは覚えておかなくてはなりません．

プログラムを書く紙のことをコーディングシートといいます．図4．2を見て下さい．これがコーディングシートの見本なのです．いろいろな文字が書いてありますね．

FORTRAN PROGRAM SHEET

TITLE: TK-85 矢印プログラム　PROGRAMMER: Y.TAKEUCHI　DATE: 1980.1.1

ADDRESS	LABEL	MNEMONIC	OPERAND	OP CODE	
8000	START	LXI	H,83F8H	21F883	
8003		MVI	A,8H	3E08	
8005		MVI	M,40H	3640	
8007		CALL	8030H	CD3080	
800A		MVI	M,0H	3600	
800C		INX	H	23	
800D		DCR	A	3D	
800E		JNZ	8005H	C20580	
8011		JMP	8000H	C30080	
8030	SUB	MVI	D,40H	1640	D←40H
8032		MVI	B,0H	0600	B←0H
8034		DCR	B	05	B←B−1
8035		JNZ	8034H	C23480	[illegible]
8038		DCR	D	15	D←D−1
8039		JNZ	8032H	C23280	[illegible]
804C		RET		C9	

図4.2

図の左上から説明しましょう．

●ADDRESS（アドレス）

8085 CPU はプログラムをメモリに入れ，それを実行していきます．そのプログラムをメモリの何番地に入れるかを示しているのがこのアドレス欄です．8085 CPU のアドレス線は16ビットですので，16進数では 4 桁の数字で表すことができます．そのため，アドレスは16進数 4 桁で書かれます．

●LABEL（ラベル）

大きな仕事も細かくみると小さな仕事の積み重ねです．その小さな仕事の見出しを書くところです．プログラムが見やすくなりますし，プログラムを書くときにも大変役立つ欄です．

●MNEMONIC（ニーモニック）

8085 CPU の命令，つまり機械語は普通16進数で表示されます．しかし，人間には16進数の並びを見てそれがどのような命令かを知るのは大変むずかしいことです．そこで，人間にとってわかりやすい記号で命令を示すところ，それがニーモニック欄です．

●OPERAND（オペランド）

先のニーモニック欄には例外もありますが「何をする」の「する」にあたる部分しかありません．その動作の主体となる「何を」を表すのがこのオペランド欄です．

●OP CODE（オペコード）

ここには 8085 CPU の命令を16進数でそのまま書いてあります．TK-85 にプログラムを入力

するときにはこの欄を見ながら入力することになります．

ここに書いたコーディングシートの例は絶対ではありません．しかし，ここに書いたことを覚えておけばどんなコーディングシートを見ても理解できると思います．

4．8085 CPU の命令体系

8085 CPU の命令解説

8085 CPU の命令は大きくわけて四つあります．

(1) 転送命令

(2) 演算命令

(3) ジャンプ・サブルーチン命令

(4) その他の命令

ではそれぞれの命令群の解説とその命令の代表例として矢印プログラムに使われている命令を解説することにします．

(1) 転送命令

転送命令とはレジスタとレジスタ間やメモリとレジスタ間での情報の移動についての命令のことです．

では，転送命令の代表について説明することにしたいのですが，その前に知っておいてもらいたい記号がいくつかあるのでそれらを先に説明しておきます．

2章で少し説明しましたが 8085 CPU 命令には1バイト命令，2バイト命令，3バイト命令の3種類があるといいました．つまり1バイトでひとつの命令を表す命令，2バイト，3バイトがひとかたまりになって命令を表す命令があるということです．この2バイト命令の2バイト目をこれ以後 B_2 と表し，同様に3バイト命令の2バイト目，3バイト目をそれぞれ B_2 B_3 と表すことにします．

それから r という文字を A (アキュムレータ)，B，C，D，E，H，L レジスタのうちの任意のものを表すとします．

また（ ）はレジスタなどの内容を表すのに使います．たとえば（D）と書けば，D レジスタの内容を表すことになります．

それから〈B_2〉，〈B_3〉はそれぞれ命令の2バイト目の内容，3バイト目の内容を表します．そして〔 〕は〔 〕の中で指示されるメモリ番地の内容を示すことにします．

たとえば〔(H)(L)〕と書けば HL レジスタペアで示されるメモリ番地の内容，〔〈B_3〉〈B_2〉〕と書けばB_3，B_2で指示されるメモリ番地の内容です．

ほかにもいろいろ説明しておきたい記号があるのですが，これくらいにして，転送命令の代表的な命令の解説をしていきましょう．

1．MVI　r，〈B_2〉
（r）　←　〈B_2〉

move immediate と呼びます．2バイト命令で，〈B_2〉を任意のレジスタに入れることができます．

MVI　B，3BH

と書けばBレジスタに 3BH というデータを入力することになります．

2．MVI　M，〈B_2〉
〔(H)(L)〕　←　〈B_2〉

move to memory immediate の略です．これも2バイト命令で〈B_2〉をH，Lレジスタで指示されるメモリ番地にしまいます．

たとえばHLレジスタペアに 83F8H というデータが入っているとき，

MVI　M，30H

と書くと，83F8H 番地に 30H というデータを格納します．

3．LXI　rp，〈B_3〉〈B_2〉
(rh)　←　〈B_3〉　　(rl)　←　〈B_2〉

load register pair immediate の略です．rp とはレジスタペアのことです．rh とはレジスタペアの上位，rl とはレジスタペアの下位といいます．HL レジスタでいえば rh は Hレジスタ，rl はLレジスタのことです．

3バイト命令で〈B_2〉と〈B_3〉がレジスタペア，BC，DE，HL，SP レジスタのどれかに入ります．

LXI　H，831FH

と書くと HL レジスタペアに 831FH が入ります．

(2)　演算命令

演算命令を細分化すると次の三つになります．

1．インクリメント・デクリメント命令
2．算術演算命令
3．論理演算命令

1．のインクリメント・デクリメント命令はレジスタやメモリの内容を＋1したり－1したりする命令です．この命令はキャリーを除くすべてのフラグを変化させます．ここでフラグの説明を少ししておきます．

フラグはある命令を実行したときにその結果の状態によって1か0かになります．

たとえば演算命令を実行したときにその結果が0だとZフラグが1，そうでなければ0．

キャリー（桁上げ：つまり答えが8ビットの最大値11111111_2を超えたときなど）がでれば CY フラグ＝1となり，そうでなければ0．最上位ビットが1ならSフラグ＝1，そうでなければ0．

8ビットのレジスタの中味の1の数が偶数のときPフラグ＝1，奇数のとき0となります．

これらのフラグは我々がフラグの結果を必要とするとき以外にも次々に変化していますがそれらは気にしないで，フラグを見る必要のあるときだけフラグを見ればよいのです．

このフラグは後で説明する条件 JUMP のときなどに使われます．

2．算術演算命令

これはA（アキュムレータ）レジスタを中心としてレジスタ間の和や差を計算します．

これによってフラグはすべて変化します．

3．論理演算命令

A（アキュムレータ）レジスタを中心にレジスタ間の論理演算（AND, OR, XORなど）を行います．フラグが変化します．

さて矢印プログラムで使用しているのは1．のインクリメント・デクリメント命令です．

それは次の二つです．

1．INX　rp

(rp)　←　(rp)＋1

increment register pair の略で，指定されたレジスタペアに＋1します．フラグに影響を与えません．

2．DCR　r

(r)　←　(r)－1

decrement register の略で任意のレジスタから－1します．フラグが変化します．

(3)　ジャンプ・サブルーチン命令

1．JMP

(PC)　←　$\langle B_3 \rangle \langle B_2 \rangle$

Jump 命令は3バイト命令です．$\langle B_3 \rangle$ がプログラムカウンタの上位に，$\langle B_2 \rangle$ が下位に入ります．この結果そのメモリ番地に CPU は実行を移します．

2．JNZ

Zフラグが0なら JMP 命令を実行します．Zフラグが1なら何もしないで次の命令に移ります．

3．CALL

Call 命令はそのときのプログラムカウンタの命令をスタックに入れて戻り番地として記憶し，スタックポインタを小さい方へ2動かし，アドレスのところで指定されている番地のところへジャンプします．サブルーチンへ飛ぶために使います．

〔(SP)－1〕　←　(PCH)

〔(SP)－2〕　←　(PCL)

(SP)　←　(SP)－2

(PC)　←　$\langle B_3 \rangle \langle B_2 \rangle$

4．RET

(PCL)　←　〔(SP)〕

(PCH)　←　〔(SP)＋1〕

(SP)　←　(SP)＋2

Return 命令でスタックの情報をプログラムカウンタに戻しサブルーチンからメインルーチンに戻ります．

(4)　その他の命令

(1)～(3)の命令群のほかには割り込み命令やI/Oポートへの命令がありますがそれについては後で学ぶことになります．

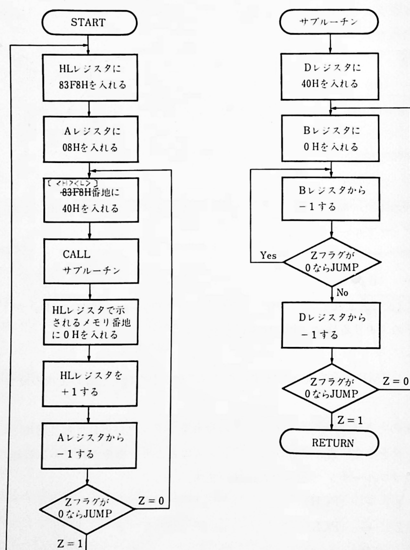


図4.3

以上大ざっぱですが 8085 CPU の命令体系を見てきました．次章以降でそれぞれの命令群の個々の命令について詳しく見ることにしましょう．

5．矢印プログラムについて

4節で覚えた命令を使えばもう1章で動かしてみた矢印プログラムを理解できます．この節では矢印プログラムを解説することにしましょう．

矢印プログラムのフローチャート

フローチャートをお見せする前に TK-85 のLED を操作するには TK-85 のメモリの 83F8H 番地から 83FFH 番地に適当な16進数2桁のデータを入力すればよいのです．

たとえば 83F8H 番地に 7FH というデータを入れると左端の LED が 8 になります．これだけを覚えておいて図4.3のフローチャートを見て下さい．

サブルーチンはウェイトループで時間つぶしのプログラムです．これを 8085 CPU のプログラムで書くと1章のリスト1.1になります．フローチャートをよく見ながらプログラムリストを見て，この矢印プログラムを解読して下さい．そして，1章で少しプログラムを改造したところがありましたが，それがどのような意味を持っているかを考えて下さい．

6．アセンブラ言語について

アセンブラとはニーモニックコードで書かれたプログラムを自動的に機械的に翻訳するプログラムのことです．

TK-85 ではこのアセンブラ言語をもっていません．ですからニーモニックコードで書かれたプログラムを「インストラクション活用表」などによって手作業で機械語，つまり16進数に変換することになります．

この作業はプログラムを作ること自体には全く関係なく，ただの機械的な作業ですのでこれらはコンピュータにまかせることができるのです．

この考え方をさらに拡張すれば，プログラムの中で sin30°と書いただけで，それを機械語のレベルまで翻訳してくれるプログラムを作ることも可能だということがわかります．このような言語は高級言語と呼ばれ，一般にはフォートラン，コボルなどの名前で知られているものがそれにあたります．

この章では 8085 CPU を使ったプログラムの作り方についての解説をしました．8085 CPUのプログラムの基礎について，おわかりいただけたと思います．次章以降では，さらにプログラムの技術をみがいていくことにしましょう．

5章　データの取り扱い

1．はじめに

本章では，まずマイクロコンピュータを使うときの基本的なデータの取り扱い方として，レジスタ間，レジスタとメモリ間のデータの転送方法について，さらにデータをメモリに記憶したり，逆にそれを引用するのに，メモリの番地を直接に指定する方法と，間接に指定する方法について述べ，最後に，データの算術演算として加算・減算を，論理演算として否定・論理積・論理和・桁移動について述べます．

2．データの転送

2.1　レジスタ間のデータの転送

まず，レジスタ間におけるデータの取り扱いとして，1バイトの定数をAレジスタに入れる場合は，たとえば MVI (move immediate) 命令を用いて

```
MVI    A, 3
```

のように書きます．このようにすると定数3がAレジスタに入ります．ここでレジスタとしては A，B，C，D，E，H，Lのいずれを用いてもかまいません．

次に特定のレジスタを他のレジスタに転送するには，たとえば，MOV (move) 命令を用いて

```
MOV    B, A
```

とします．このようにしますとAレジスタの内容がBレジスタに転送されます．このときAレジスタの内容は変化しません．

例5－1　定数23をBレジスタに入れ，それをCレジスタに転送するプログラムは右のようになります．	MVI　B，23 MOV　C，B HLT

このようにプログラムの最後には必ず一時停止命令 HLT を書いておきます．

上に述べた二つの命令は1バイトのデータしか取り扱うことができません．すなわち10進数の0から255までしか扱えません．それ以上のデータは2バイトを用いて表現する必要があります．2バイトのデータは二つのレジスタを対にして取り扱いますが，対にできるレジスタはきめられていて，BとC，DとE，およびHとLレジスタが対にして用いることができます．たとえば10進数の定数300を HL レジスタ対に入れる場合には LXI (load register pair immediate) 命令を用いて

LXI　　H, 300

と書き，また16進数の定数 ABCD を HL レジスタ対に入れるには

LXI　　H, 0ABCDH

と書きます．このときHレジスタに16進数の上2桁のAB，Lレジスタには下2桁の CD が入ります．これらの処理条件に書いた H は HL レジスタ対を表しており，H を B，D，SP と書きかえると，2バイトの定数がそれぞれ BC，DE レジスタ対またはスタックポインタに入ります．

二つのレジスタ対にある2バイトのデータを交換するために XCHG (exchange HL with DE) 命令

XCHG

があります．この命令を実行すると HL レジスタ対の内容と DE レジスタ対の内容が入れかわります．

2．2　番地の直接指定によるデータの記憶と引用

次にレジスタにあるデータをメモリに記憶する場合を考えてみましょう．それには記憶するメモリの番地を直接に指定する方法と，それを間接に指定する方法とがあります．ここでは前者について述べ，後者は2．3項で述べることにします．

まずAレジスタに保持している1バイトの定数を8100H番地のメモリに転送して記憶するためには，STA (store accumulator direct) 命令を用いて

STA　　8100H

と書きます．

例5－2　定数5を8100H番地のメモリに記憶するプログラムは右のようになります．	MVI　A，5 STA　8100H HLT

逆に，メモリに記憶してある定数をレジスタに転送するには，たとえば，LDA (load accumulater direct) 命令を用いて

LDA　　8100H

と書きます．このようにすると8100H番地のメモリにある定数がAレジスタに転送されます．

例5－3　8100H番地のメモリに定数5を記憶してから，それを8200H番地に転送するプログラムは右のようになります．	MVI　A，5 STA　8100H LDA　8100H STA　8200H HLT

次に2バイトの定数を連続する二つの1バイトのメモリに記憶するためには，まずその定数を HL レジスタ対に入れてから SHLD (store HL direct) 命令を用いて

8100H ← L
8101H ← H

SHLD　8100H

と書きます．このとき図５．１のようにＬレジスタにある定数の部分が8100Ｈ番地のメモリに入り，Ｈレジスタにある部分が8101Ｈ番地のメモリに記憶されます．

例５－４　定数300を8100Ｈ番地と8101Ｈ番地を対にしたメモリに記憶するプログラムは右のようになります．	LXI　H，300 SHLD　8100H HLT

３バイト以上のデータの場合は，２バイトずつに区切り，上の手順を繰り返し用いることができます．

例５－５　16進数の定数123456ABを8100Ｈ番地から始まる連続する４バイトのメモリに記憶するプログラムは右のようになります．	LXI　H，1234H SHLD　8100H LXI　H，56ABH SHLD　8102H HLT

このプログラムを実行するとHLレジスタ対はまず［12｜34］となり，次に［56｜AB］となります．この結果メモリには図５．２のように記憶されます．

逆に，連続する２バイトのメモリに記憶してある定数をHLレジスタ対に転送するには，たとえばLHLD（load HL direct）命令を用いて

LHLD　8100H

とします．このとき8100Ｈ番地と8101Ｈ番地を対にしたメモリに記憶してある２バイトの定数が，HLレジスタ対に転送されます．

例５－６　8100Ｈ番地，8101Ｈ番地のメモリを対にして２バイトの定数12ＡＢＨを記憶してから，それを8200Ｈ，8201Ｈ番地を対にしたメモリに転送して記憶するプログラムは右のようになります．	LXI　H，12ABH SHLD　8100H LHLD　8100H SHLD　8200H HLT

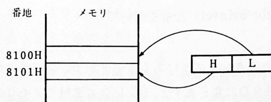


図5.1　命令SHLDを用いてHLレジスタ対にあるデータをメモリに転送する

番地	メモリ
8100H	34
8101H	12
8102H	AB
8103H	56

図5.2　例5-5を実行したときのメモリの様子

2.3　番地の間接指定によるデータの記憶

データをメモリに記憶するのに，まずそのメモリの番地を一度HLレジスタ対に入れてから，それを引用して記憶することもできます．たとえば，HLレジスタ対で指定したメモリの番地(M)に定数3を記憶するには，

```
MVI     M, 3
```

と書きます．

例5－7　間接指定の手法を用いて定数5を8100H番地に記憶するプログラムは右のようになります．	LXI　H, 8100H MVI　M, 5 HLT

このプログラムは例5-2と同じ働きをします．

例5－8　8100H，8101H番地のメモリに81A0H番地を記憶してから，その81A0H番地のメモリに定数ABHを記憶するプログラムは右のようになります．	LXI　81A0H SHLD　8100H LHLD　8100H MVI　M, 0ABH HLT

レジスタに入っているデータをHLレジスタ対に保持している番地のメモリに記憶するときには

```
MOV     M, A
```

と書きます．このときAレジスタに入っているデータがHLレジスタ対に保持している番地のメモリに記憶されます．この処理条件のAは他のレジスタ名でもかまいません．

例5－9　定数5をBレジスタに入れてから8100H番地に記憶するプログラムは右のようになります．	MVI　B, 5 LXI　H, 8100H MOV　M, B HLT

間接指定の手法を用いてAレジスタのデータをメモリに記憶したいときに，メモリの番地を指定するHLレジスタ対が使えない場合は，たとえばBCレジスタ対に記憶するメモリの番地をあらかじめ入れてから，STAX (store accumulator indirect) 命令を用いて

```
STAX    B
```

のように書くことができます．このようにしますと，AレジスタにあるデータがBCレジスタ対にある番地のメモリに記憶されます．処理条件のBをDに変えますと，DCレジスタ対でメモリの番地を指定することができます．

例5－10　定数5をAレジスタに入れてから，8100H番地に記憶させるプログラムを，HLレジスタ対を用いないで書きますと右のようになります．	MVI　A，5 LXI　B，8100H STAX　B HLT

このプログラムの第2行目と第3行目の処理条件のBをDに変えてもかまいません．

2．4　番地の間接指定によるデータの引用

メモリに記憶してあるデータを引用するときに，LDA 命令を用いてその番地にあるデータを直接引用しないで，いったんその番地を HL レジスタ対に入れてから，それを用いてAレジスタにデータを転送するには

MOV　　A，M

と書きます．このとき HL レジスタ対に入れた番地のメモリの内容がAレジスタに転送されます．処理条件のAは他のレジスタ名でもかまいません．

例5－11　あらかじめ8100H番地に定数3を記憶しておき，それをBレジスタに転送するプログラムは右のようになります．	MVI　A，3 STA　8100H LXI　H，8100H MOV　B，M HLT

HL レジスタ対が使えない場合，メモリに記憶してあるデータを間接指定の手法によってAレジスタに転送するには，たとえば BC レジスタ対に記憶するメモリの番地をあらかじめ入れてから LDAX (load accumulator indirect) 命令を用いて

LDAX　　B

と書きます．このとき BC レジスタ対にある番地のメモリの内容がAレジスタに転送されます．この処理条件のBをDに変えることによりメモリの番地を指定するのに DC レジスタ対が使えます．

例5－12　あらかじめ8100H番地に定数3を記憶しておき，それをAレジスタに転送するプログラムをHLレジスタ対を用いないで書きますと右のようになります．	MVI　A，3 STA　8100H LXI　B，8100H LDAX　B HLT

このプログラムの第3行目と第4行目のBをDに変えてもかまいません．

最後にこの節で説明しましたデータの転送命令を分類すると，定数をレジスタまたはメモリに入れる場合，レジスタにある内容を他のレジスタに転送する場合，レジスタの内容をメモリへ，またはその逆にメモリの内容をレジスタへ転送する場合とがあります．これらの各場合の使用例

表5．1　データの転送命令の使用例

分　類	使　用　例	動　作	注　意
定数 → レジスタ	MVI　A，3	3 → A	（注1）
	LXI　H，300	300 → HL	2バイトの定数（注2）
定数 → メモリ	MVI　M，3	3 →（HL）	
レジスタ → レジスタ	MOV　B，A XCHG	A → B HL ↔ DE	（注3）
レジスタ → メモリ	STA　8100H MOV　M，A STAX　B SHLD　8100H	A →（8100H） A →（HL） A →（BC） H →（8101H），L →（8100H）	（注4） （注1） （注5） 2バイトの定数（注4）
メモリ → レジスタ	LDA　8100H MOV　A，M LDAX　B LHLD　8100H	（8100H）→ A （HL）→ A （BC）→ A （8101H）→ H，（8100H）→ L	（注4） （注1） （注5） 2バイトの定数（注4）

（注1）　AはB，C，D，E，H，Lでもよい．
（注2）　HはB，D，SPでもよい．
（注3）　B，AはA，B，C，D，E，H，Lのいずれでもよい．
（注4）　処理条件は2バイトの番地．
（注5）　BはDでもよい．

を表5．1にまとめておきます．

3．　データの算術演算

マイクロコンピュータで実行できるデータの演算としては，算術演算と論理演算の2種類があります．前者にはここで述べる加算と減算があり，後者には次節で述べる否定，論理積，論理和，桁移動などがあります．この他の演算は，これらの演算を組み合わせて実行します．

3．1　加　　算

2数の加算を実行する加算命令には以下で述べますように6種類ありますが，いずれの場合も加算結果はAレジスタに入ります．

まず，Aレジスタの内容に1を加えるときはINR（increment）命令を用いて

```
INR    A
```

と書きます．この命令の処理条件にはどのレジスタ名を書いてもよく，そのレジスタの内容に1が加わります．またメモリに記憶してあるデータに1を加える場合は，まずそのメモリの番地をHLレジスタ対に入れてから

```
INR    M
```

と書きます．

レジスタ対の内容に1を加える場合はINX命令を用います．たとえばINX(increment reister pair)命令を用いて，

INX　　H

と書きますとHLレジスタの内容に1が加算されます．この処理条件はB，DまたはSPと書いてもよく，そのときはそれぞれBC, DEレジスタ対またはスタックポインタの内容に1が加算されます．

例5－13　81A0H番地から81A2H番地までのメモリに10から12までの定数を記憶するプログラムは右のようになります．	MVI STA INR STA INR STA HLT	A，10 81A0H A 81A1H A 81A2H

次に，Aレジスタにある数値と1バイトの定数を加えて，それをAレジスタに入れるにはADI (add immediate) 命令を用います．たとえば，Aレジスタの値に3を加えてAレジスタに入れるには

ADI　　3

と書きます．また加える数がレジスタに前もって入っている場合には，ADD(add)命令を用います．たとえばAレジスタとBレジスタにある二つの数値を加えてAレジスタに入れるときは，

ADD　　B

と書けばよいのです．この命令の処理条件としてはA．B，C，D，E，H，Lのいずれのレジスタ名を書いてもかまいません．なお，処理条件にAを書くとAレジスタの値を2倍することになります．さらにADI, ADD命令の代りに，ACI, ADC命令を用いますと，上記の結果にさらに最上位のビットから桁上げがあったときに生じるCYフラグの値も加算されます．これは2バイトの数値の加算で桁上げがあるときなどに使用します．

例5－14　3＋2を計算して8100H番地へ記憶するプログラムは右のようになります．	MVI ADI STA HLT	A，3 2 8100H

次に，Aレジスタのデータとメモリに記憶してあるデータを加えるときには，あらかじめそのメモリの番地をHLレジスタ対に入れてから

ADD　　M

と書きます．さらに，ここにCYフラグの値も加えるときには，このADD命令の代りにADC命令を用います．

例5－15　2バイトの定数があり下位のバイトが8100H番地のメモリに，上位のバイトが8101H番地のメモリに記憶してあります．同様に8200H番地と8201H番地にも2バイトの定数が入っています．これらの定数を加えて，その結果を82ABH番地と82ACH番地に入れるプログラムは右のようになります．	LXI　B，8100H LXI　D，82ABH LXI　H，8200H LDAX　B ADD　M STAX　D INX　B INX　D INX　H LDAX　B ADC　M STAX　D HLT

このプログラムでは2バイトの定数を上下のバイトに分けて，1バイト命令を用いて加算を行っています．すなわち，第6行目の STAX 命令までで下位8ビットの加算を行い，第7行目から上位8ビットの加算を行っています．このとき下位の加算で生じた桁上げを加えるために第11行目の ADC 命令を用いています．

最後に二つの2バイトのデータを直接加算できる DAD (doubre register add) 命令があります．まず一方のデータを HL レジスタ対に入れ，他方のデータを他のレジスタ対に入れてから DAD 命令を実行します．たとえば，HL レジスタ対と BC レジスタ対にある2バイトのデータを加算するときは DAD 命令を用いて

DAD　B

と書きます．このとき結果は HL レジスタ対に入ります．この命令の処理条件にはBの他D，H，SP と書いてもかまいません．

例5－16　2バイトの定数があり，下位のバイトが8100H番地のメモリに，上位のバイトが8101H番地のメモリに記憶してあります．それを2桁左へ移動するプログラムは右のようになります．	LHLD　8100H DAD　H DAD　H SHLD　8100H HLT

このプログラムは8100H番地のメモリに記憶してある値に，その値を加算し，さらにその結果を再度加することによって2桁左へ移動しています．

例5－17　2バイトの定数があり，下位のバイトが8100H番地のメモリに，上位のバイトが8101H番地のメモリに記憶してあります．この定数を5倍し，その結果を連続する82EEH，82EFH番地のメモリに記憶するプログラムは右のようになります．	LHLD　8100H MOV　C，L MOV　B，H DAD　H DAD　H DAD　B SHLD　82EEH HLT

このプログラムではHLとBCレジスタ対に入れた定数を第4，5桁目のDAD命令で4倍しています．第6行目のDAD命令でその定数を4倍した結果に，再びその定数を加えていますので，結局5倍したことになります．

例5－18　DAD命令を用いて例5－15を書き換えますと右のようになります．	LHLD　8100H XCHG LHLD　8200H DAD　D SHLD　82ABH HLT

今までに述べてきましたすべての加算命令をその使用例とともに表5．2に示しておきます．

表5．2　加算命令の使用例．ただし(　)内の命令を用いるときには，さらにCYフラグの値も加算される．

分類	使用例	動作	注意
＋1	INR A INR M INX H	A＋1 → A (HL)＋1 →(HL) HL＋1 → HL	(注1) (注2)
加算	ADI(ACI) 3 ADD(ADC) B ADD(ADC) M	A＋3 → A A＋B → A A＋(HL) → A	(注1)
2バイトの加算	DAD B	HC＋BC → HL	(注2)

(注1)　処理条件はどのレジスタ名でもよい．
(注2)　処理条件はB，D，H，SPのいずれでもよい．

3．2　減　算

加算命令と同様に，減算命令も6種類あり，それらの減算結果はいずれもAレジスタに入ります．

Aレジスタの内容から1を引くときはDCR (decrement) 命令を用いて

DCR　A

と書きます．この処理条件にはどのレジスタ名を書いてもかまいません．またメモリに記憶して

あるデータから１を引く場合は，まずそのメモリの番地を HL レジスタ対に入れてから

```
DCR     M
```

と書きます．

レジスタ対の内容から１を引く場合は DCX（decrement register pair）命令を用います．たとえば，

```
DCX     H
```

のように書きますと，HL レジスタ対の内容から１が引かれます．この処理条件はB，Dまたは SP と書いてもかまいません．

減算命令にはAレジスタの内容から定数を引く SUI（subtract immediate）命令，Aレジスタの内容からレジスタやメモリにある内容を引く SUB（subtract）命令があります．これらは加算命令と同様にして使用できます．たとえば，Aレジスタの内容から定数３を引いてその結果をAレジスタに入れるには，

```
SUI     3
```

と書き，またAレジスタの内容からBレジスタの内容を引き，その結果をAレジスタに入れるには

```
SUB     B
```

と書きます．この処理条件には他のレジスタ名を書いてもよく，またAレジスタの内容からメモリの内容を引く場合は，あらかじめそのメモリの番地を HL レジスタ対に入れてから SUB 命令の処理条件をMと書きます．これらの使用例は表５．３にまとめておきます．

表５．３　減算命令の使用例．ただし（　）内の命令を用いるときには，さらにＣＹフラグの値も減算される．

分　類	使　用　例	動　　作	注　意
－　１	ＤＣＲ Ａ ＤＣＲ Ｍ ＤＣＸ Ｈ	Ａ－１ → Ａ （ＨＬ）－１ →（ＨＬ） ＨＬ－１ → ＨＬ	（注１） （注２）
減　算	ＳＵＩ（ＳＢＩ） ３ ＳＵＢ（ＳＢＢ） Ｂ ＳＵＢ（ＳＢＢ） Ｍ	Ａ－３ → Ａ Ａ－Ｂ → Ａ Ａ－（ＨＬ）→ Ａ	（注１）

（注１）　処理条件はどのレジスタ名でもよい．
（注２）　処理条件はＢ，Ｄ，Ｈ，ＳＰのいずれでもよい．

例５－19　82ＡＢＨ番地のメモリに負の定数を記憶しておき，それを正になおして82ＡＣＨ番地に記憶するプログラムは右のようになります．

注）便宜上データを２進数で書く場合，小さな添字 ２ をつけておきますが，プログラムのコーディング時にはこの書き方はできません．

```
MVI     A, 11111011₂
STA     82ABH
LXI     H, 82ABH
MOV     B, M
MVI     A, 0
SUB     B
STA     82ACH
HLT
```

3．3　2進化10進数による加算

いままで述べてきました命令の処理条件に書かれた定数は，プログラムを機械語になおして実行するときに，16進数でキーボードから入力する必要があります．また内部での処理は2進数で行われますが，その結果を外部に出力するときは1バイトを4ビットずつ区切って2桁の16進数の形で表示されます．そこでこの16進数を10進数に直す必要があります．この場合大きい数値の場合は，このような入力時と出力時における変換がめんどうになります．

加算の場合，この不便さをさける手法があります．まず16進数のキーボードを10進数のキーボードと見なしてそのまま入力しますと2進化と10進数の形で入力できます．二つの数をこのようにして入力したものを加算した結果をもう一度正しい2進化10進数に修正する必要があります．このために DAA（decimal adjust accumulator）命令

DAA

用います．

たとえば11＋9＝20の加算を行うとき，16進のキーボードで入力しますと 0BH，09H と16進数になおさなくてはいけませんが，2進化10進数で加算する場合は，11と9をそのまま入力しますと，それぞれ00010001_2，00001001_2となって，2進化10進数で入力できます．しかしこの二つの数値を加算すると

$$00010001_2 + 00001001_2 = 00011010_2$$

となり，結果の下位4ビットが9を超えるので，2進化10進数になるように補正する必要があります．このとき DAA 命令を用いると，00011010_2が自動的に00100000_2に補正されます．

例5－20　11と9を2進化10進数で入力し，その和を求めて8100H番地のメモリに記憶するプログラムは右のようになります．	MVI　A，11H MVI　B，9 ADD　B DAA STA　8100H HLT

この例では和が0から99までの10進数になる場合の加算しかできませんが，少し工夫すると大きな10進数の加算ができます．

例5－21　8100H番地と8101H番地のメモリに2進化10進数で記憶してある4桁の10進数の定数と，同じく8102H番地と8103H番地のメモリに記憶してある4桁の10進数の加算を行ない，その結果を2進化10進数で8104H番地と8105H番地のメモリに記憶するプログラムは右のようになります．ただし，8100H，8102H，8104H番地のメモリには下位の桁を記憶するものとします．

```
LXI     B, 8100H
LXI     H, 8102H
LXI     D, 8104H
LDAX    B
ADD     M
DAA
STAX    D
INX     B
INX     H
INX     D
LDAX    B
ADC     M
DAA
STAX    D
HLT
```

このプログラムでは，まず第4行目と第5行目で10進数の下位2桁の加算を行っています．第6行目のDAA命令で和を正しい10進数に修正をしてから，第7行目のSTAX命令で8104H番地のメモリに記憶しています．同様に上位の2桁の加算を第12行目のADC命令で行っています．このときは下位の桁から桁上げも加算しています．このプログラムで，たとえば1234＋5678を行ないますと，

番地	メモリ
8100H	34
8101H	12
8102H	78
8103H	56
8104H	12
8105H	69

のように記憶されます．

4．論理演算

4.1 否　定

Aレジスタにある1バイトのデータの各ビットの否定をとるときには　CMA（complement accumulator）命令を用いて

```
CMA
```

と書きます．この命令を実行するとAレジスタにある8ビットの数値の1と0が入れ換わります．

例5－22　Aレジスタに定数5を入れて，その2の補数を求めてBレジスタに転送するプログラムは右のようになります．	MVI　A，5 CMA INR　A MOV　B，A HLT

補数は各ビットの1と0を入れ換えて，その結果の最下位に1を加えれば求まりますのでCMA命令とINR命令を用いて実行しています．その結果Bレジスタには11111011_2が入ります．

一方，CYフラグに1を入れるときはSTC（set carry）命令を用いて

STC

と書き，CYフラグの値の否定をとるときにはCMC（complement carry）命令を用いて

CMC

と書きます．このときレジスタやCY以外のフラグの値は変わりません．

例5－23　CYフラグの値を0にするプログラムは右のようになります．	STC CMC HLT

あらかじめSTC命令でCYフラグの値を1にし，それをCMC命令によって反転するとCYフラグに0が入ります．

4．2　論　理　積

値が0か1をもつ二つの1ビットの変数x_1，x_2から論理積x_1 and x_2，論理和x_1 or x_2，排他的論理和x_1 xor x_2を求めると表5．4のようになります．1バイトの二つのデータの対応する各ビットごとに，すべてのビットについて表5．4の論理積，論理和，排他的論理和の演算を行う命令があります．これらを用いるときは一方のデータはAレジスタに入れておく必要があり，また演算結果はAレジスタに入ります．これらの論理演算命令にはAレジスタの内容と定数，他のレジスタまたはメモリの内容との間で演算を行うかによって，それぞれ3種類の命令があります．

表5．4　2変数の論理演算

x_1	x_2	x_1 and x_2	x_1 or x_2	x_1 xor x_2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

たとえば，Aレジスタの内容と定数3Hとの各ビットの論理積を求めて，その結果をAレジスタに入れるためにはANI（and immediate）命令を用いて

ANI　　3H

とします．またAレジスタの内容とBレジスタの内容との間で論理積を求め，その結果をAレジスタに入れるためにはANA（and）命令を用いて

ANA　　B

とします．この処理条件はA以外のどのレジスタを用いてもかまいません．また，あらかじめHLレジスタ対でメモリの番地を指定しておき，その番地のメモリにある内容とAレジスタの内容の論理積を求めるときには命令ANAの処理条件をMとすればよいのです．

例5−24　8100H番地にある2進数の定数が負の場合には，Aレジスタのビット7の値だけを1にするプログラムは右のようになります．	LDA　8100H ANI　10000000_2 HLT

4.3　論　理　和

論理和，排他的論理和の場合も同様で，Aレジスタの内容と定数との間で演算を行うためには，それぞれ，たとえば

ORI　　3

XRI　　3

のように，定数を処理条件に書いたORI（or immediate），XRI（exclusive or immediate）命令を用います．またAレジスタの内容とレジスタまたはメモリの内容との間で論理和の演算を行うためには，ORA（or）命令を用いて

ORA　　B

と書きます．また排他的論理和の場合はXRA（exclusive or）命令を

XRA　　B

表5.5　論理演算命令の使用例

分　類	使　用　例	動　作	変化するフラグ
論理積	ANI　3 ANA　B ANA　M	A and 3 → A A and B → A A and M → A	Z, S, P CY, AC → 0
論理和	ORI　3 ORA　B ORA　M	A or 3 → A A or B → A A or M → A	
排他的論理和	XRI　3 XRA　B XRA　M	A xor 3 → A A xor B → A A xor M → A	

のように書いて用います．ただし，Mと書いてメモリの内容と演算するときにはあらかじめHLレジスタ対にそのメモリの番地を入れておく必要があります．

最後に，いままで述べた論理積，論理和，排他的論理和命令とその使用例を表5．5にまとめておきます．ここで変化するフラグの項については次節で述べます．

例5－25　8100H番地にある2進数の定数が偶数なら，Aレジスタのビット0を1にすることによって奇数になおすプログラムは右のようになります．	<pre>LDA 8100H ORI 1 HLT</pre>

例5－26　2進数の二つの定数11001001_2と00001111_2の論理積，論理和，排他的論理和をとり，それぞれの結果を8100H番地から連続する三つのメモリに記憶するプログラムは右のようになります．	<pre>MVI B，11001001₂ MVI C，00001111₂ MOV A，B ；AND ANA C STA 8100H MOV A，B ；OR ORA C STA 8101H MOV A，B ；XOR XRA C STA 8102H HLT</pre>

このプログラムでは，まず二つの定数をB，Cレジスタに入れて，それぞれの論理演算をするときに，BレジスタからAレジスタにデータを移して用い，Bレジスタのデータがこわれないようにしています．

表5.5の論理計算命令を繰り返し用いて二つ以上の定数に対する論理演算を行うこともできます．

例5－27　2進数の三つの定数11001001_2と00001111_2と10101010_2の論理積をとり，結果を8100H番地に記憶するプログラムは右のようになります．	<pre>MVI A，11001001₂ MVI B，00001111₂ MVI C，10101010₂ ANA B ANA C STA 8100H HLT</pre>

4.4 桁　移　動

Aレジスタの各ビットの値を順次左へ移動するときに用いられる命令としてRLC (rotate accumulator left) 命令

RLC

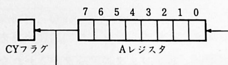
とRAL (rotate left through carry) 命令

RAL

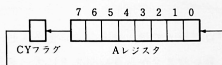
の二つがあります．図5.3のようにRLCを実行すると，ビット7がビット0に直接移動します．このとき，ビット7の値がCYフラグにも入ります．このRLC命令をm回用いてCYフラグの値が常に0ならばAレジスタの内容を2^m倍することができます．一方，RAL命令の場合はCYフラグも含めて環状に連結されて，それらの各ビットの値が左へ移動します．

例5－28　ビット0から順次0が入るようにして11111111_2を2回左へ移動し，その結果を8100H番地のメモリに記憶するプログラムは右のようになります．

```
MVI   A, 11111111₂
STC
CMC
RAL
CMC
RAL
STA   8100H
HLT
```



(a) RLC命令



(b) RAL命令

図5.3　左シフト命令によるデータの動き

また，このプログラムはANI命令を用いると次のように書き直すこともできます．

```
MVI   A, 11111111₂
RLC
ANI   11111110₂
RLC
ANI   11111110₂
STA   8100H
HLT
```

例5－29　定数5を10倍して，その結果を8100H番地のメモリに記憶するプログラムは右のようになります．

```
MVI    A, 5
MOV    B, A
RLC
RLC
ADD    B
RLC
STA    8100H
HLT
```

Aレジスタの数値を10倍するためには，はじめにRLC命令を2回用いて4倍し，もとの数値を加えると5倍したことになります．次にもう一度RLC命令を用いると10倍することができます．

このようにAレジスタの値を2^m以外の定数倍するときには，RLC命令とADD命令を組み合わせて実行することができます．

次に，Aレジスタの内容を右へ1ビット移動する命令としてRRC(rotate accumulator right)命令とRAR（rotate right through carry）命令の二つがあります．これらの機能は図5.4に示したように，RRC命令を実行すると，Aレジスタの値が1ビットだけ右へ移動します．このときビット0の値はビット7とCYフラグに入ります．一方，RAR命令はAレジスタにCYフラグを連結させて，それらの各ビットの値が全体として右向きに1ビット移動します．CYフラグの値を0としながらこのRAR命令をm回用いますと，Aレジスタの内容を$\frac{1}{2^m}$倍することができます．なお，定数が割りきれなくて余りがでる場合は，それを無視します．

例5－30　定数11111110_2を4で割った結果を8100H番地のメモリに記憶するプログラムは右のようになります．

```
MVI    A, 11111110₂
RRC
ANI    01111111₂
RRC
ANI    01111111₂
STA    8100H
HLT
```

2回右へ移動し，ビット7，6に数値0を入れています．

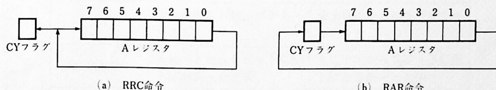


図5.4　右シフト命令によるデータの移動

6章　制　御　文

1．はじめに

本章では演算結果がどのような状態であるかを判断して，それによって次に実行するプログラムの流れを決定する制御文について述べます．まずデータの状態を判断して表示するフラグの機能について述べ，次にそれによってプログラムの実行の流れを変える方法について述べます．さらにこれらの結果を用いて，特定の手段を繰り返し実行する方法を述べます．

2．演算結果の判断

演算命令を実行して，その結果によって次に実行する文を決定したいことがあります．この場合，演算結果がどのような状態になっているかを表示するために8085では

Z，　S，　CY，　AC，　P

の5種類のフラグがあります．これらのフラグは図6.1に示すように状態レジスタの特定のビットを用いて表しています．各フラグの値は，強制的に設定するかまたは演算命令を実行したときに，自動的に表6.1のように1または0に設定されます．

5章の表5.2の加算命令，表5.3の減算命令および表5.5の論理演算命令を実行したときには，フラグの値は自動的に設定されます．たとえば

```
MVI     A, 1
ADI     2
```

S	Z	0	AC	0	P	I	CY

図6.1　状態レジスタの各ビットとフラグの対応

表6.1　フラグの値の設定

フラグ ＼ フラグの値	1	0
Z	演算結果が0のとき	演算結果が0でないとき
S	最上位の桁が1のとき	最上位の桁が1でないとき
CY	最上位桁から桁上げがあったとき	最上位桁から桁上げがなかったとき
AC	4ビットから5ビットに桁上げがあったとき	4ビットから5ビットに桁上げがなかったとき
P	8ビットの中に1が偶数個あるとき	8ビットの中に1が奇数個あるとき

として，ADI 命令によって１＋２を計算するとフラグＺ，Ｓ，CY，AC，Ｐはそれぞれ０，０，０，０，１となります．

ある状態を調べるのに表5.3の減算命令を用いてフラグの値を設定しますと，Ａレジスタの内容が変化してしまいます．減算結果は必要ありませんが，減算したときのフラグの値のみを用いたときには SUI やSUB 命令の代りに，CPI（compare immediately）命令や CMP（compare）命令を用います．

たとえば

```
CPI     3
```

としますと，Ａレジスタの内容から３を引いたときの結果によってＺ，Ｓ，Ｐ，CY，AC フラグの値を設定しますが，減算結果はＡレジスタに入らないのでＡレジスタの値はそのままです．また，Ａレジスタの内容から他のレジスタやメモリの内容を減算したときの結果によってフラグのみを変化させるには，たとえば

```
CMP     B
```

と書きます．この処理条件のＢは他のレジスタ名やＭと書いてもかまいません．ただし，Ｍと書いてメモリの内容と比較するときには，あらかじめそのメモリの番地をHL レジスタ対に入れておく必要があります．

例６－１　ＣＰＩ命令によってＣＹフラグの値を設定する手法を用いて例５－30を書き換えると右のようになります．

```
MVI     A, 11111110₂
CPI     0
RAL
CPI     0
RAL
STA     8100H
HLT
```

このプログラムでは CPI 命令を用いて CY フラグの値を０にしてから，RAL 命令によって左へ桁移動を行っています．

次にフラグの値を調べる方法を考えてみましょう．そのためには，状態レジスタの内容をみます．状態レジスタは他のレジスタのようにその内容を単独でメモリに転送することはできません．そこで状態レジスタとＡレジスタを

Ａレジスタ	状態レジスタ

のように対にして，PUSH 命令を用いて

```
PUSH    PSW
```

と書きますと，この２バイトのデータがスタックと呼ばれる補助メモリに転送されます．通常スタックの番地を保持する SP は初期値として，特定の番地を保持しています．TK-85 の場合は初期値は 8391H です．TK-85 でPUSH 命令を実行すると SP の値から２を引いた 838FH 番地の

メモリに状態レジスタの内容が入り，1を引いた 8390H 番地のメモリにAレジスタの内容が入ります．この様子を図6.2に示します．このように，状態レジスタの内容は 838FH 番地に入りますので，その内容を調べるとすべてのフラグの値がわかります．なお転送前の SP の値を任意に指定したいときは，あらかじめその値を SP に与えておく必要があります．たとえば8100H番地に状態レジスタの内容を入れたいときには

```
LXI     SP, 8101H
PUSH    A
```

とすればよいのです．

PUSH命令の処理条件としてAの外にB，D，Hを用いることができます．このときはそれぞれ BC，DE，HL レジスタ対の内容がスタックに入ります．これは後述するように，サブルーチンを用いるときにレジスタの内容を一時保護するときなどに用いられます．

例6－2　1＋2の計算をした結果，現れたフラグの値を調べるプログラムは右のようになります．	MVI　A，1 ADI　2 LXI　SP，8102H PUSH　A HLT

ここで，8101H番地の内容を調べればよいわけです．

PUSH 命令を用いるとフラグとAレジスタの内容を SP で指定したメモリへ転送することができますが，逆にこのメモリの内容をレジスタ対にもどす場合，たとえば

```
POP     A
```

と書きます．このとき図6.3に示したように SP であらかじめ指定した番地のメモリの内容が状態

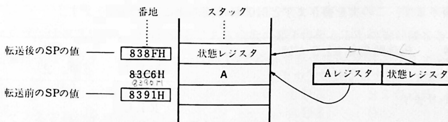


図6.2　PUSH 命令によって状態レジスタとAレジスタの内容をスタックへ転送する例

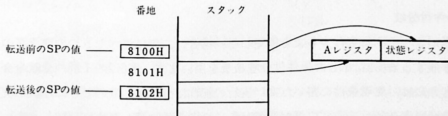


図6.3　POP 命令によってスタックからレジスタ対へ転送する例

レジスタに，その次の番地のメモリの内容がAレジスタに入ります．このときSPの値は2が加算されます．したがって状態レジスタに適当な値を入れてフラグを任意に設定したいときは，あらかじめその値をSPで指定する次の番地のメモリに入れておきます．

POP命令の処理条件としてB，D，Hを用いてもよく，このとき，スタックの内容はそれぞれBC, DE, HLレジスタ対に転送されます．このようにPUSH命令とPOP命令を用いることによって，任意の2進数をスタックに一度入れてから，それを状態レジスタに転送することができます．

例6－3　すべてのフラグの値を0にするプログラムは右のようになります．

```
MVI    H, 00000010₂
PUSH   H
POP    A
HLT
```

まず第1行目のMVIで2進数00000010_2をHレジスタに入れ，それを第2行目のPUSH命令によって現在のSPで指定されている位置から1を引いた番地のスタックに入れます．次に第3行目のPOP命令によって，それを状態レジスタに転送しています．

3．実行流れの分岐

3.1 無条件分岐

プログラムは通常書かれた命令の順に実行されますが，途中である番地の文に分岐したい場合には無条件分岐命令JMP（jump）を用いて

　　JMP　　8100H

のように書きます．この文を書きますと8100H番地の文に実行が移ります．

例6－4　桁移動命令RLCだけを実行させない例として右のプログラムがあります．

```
      MVI   A, 3
      MVI   B, 2
      JMP   JP
      RLC
JP:   ADD   B
      HLT
```

3.2 条件付分岐

演算結果によって次に実行する文を指定したい場合は，先に述べた五つのフラグのうち四つのフラグを参照する表6.2に示した条件付分岐命令を用います．表6.2のI群の分岐命令はフラグの値が1のときには，処理条件に書いた文に実行の流れは分岐し，0のときには次の文を実行します．II群の分岐命令は，フラグの値が0のときに処理条件に書いた文に分岐します．

たとえば，演算の結果，Zフラグが0になったとき，JNZ（jump non-zero)を用いて

表６．２　フラグの値によって分岐する条件付分岐命令

分岐命令		参照するフラグ	フラグの値が１のとき	フラグの値が０のとき
I	JZ JC JPE JM	Z CY P S	処理条件に書いたラベルの文へ	次の文へ
II	JNZ JNC JPO JP	Z CY P S	次の文へ	処理条件に書いたラベルの文へ

```
        JNZ    NEXT
         ⋮
NEXT:   ——
```

を実行しますと，JNZ 命令の次にラベルが NEXT の文に実行が移ります．

例６－５　あらかじめ8100H番地に記憶してある定数が偶数ならそれを8101H番地に，奇数なら8102H番地に記憶するプログラムは右のようになります．

```
        LDA    8100H
        MOV    B, A
        ANI    1
        JNZ    ODD
        MOV    A, B
        STA    8101H
        HLT
ODD:    MOV    A, B
        STA    8102H
        HLT
```

このプログラムは第３行目の ANI 命令である定数の最下位桁と１との論理積をとり，定数が奇数なら結果は１になりますので，第４行目の JNZ 命令でラベルがODD の文まで分岐します．

例６－６　あらかじめ8100H番地に記憶してある定数の絶対値をとり，その結果を8101H番地に記憶するプログラムは右のようになります．

```
        LDA    8100H
        ADI    0
        JP     PLUS
        MOV    B, A
        MVI    A, 0
        SUB    B
PLUS:   STA    8101H
        HLT
```

データの転送命令を実行してもフラグは変化しませんので，第２行目の ADI 命令でAレジスタの内容に０を加算してフラグを変化させてから，第３行目の JP 命令で符号ビットが０または正ならラベルがPLUS の文へ分岐し，符号ビットが１または負ならば，第４行目から第６行目で０

からBレジスタに入れた定数を引いて符号を反転しています.

3.3 3方向以上への分岐

演算結果によって2種類以上の分岐命令を用いますと，プログラムの流れを3方向以上に分岐させることもできます.

例6－7　8100H番地のメモリに記憶してあるデータが正なら8101H番地，負なら8102H番地のメモリにそのデータを記憶し，0なら8103H番地のメモリにFCHを記憶するプログラムは右のようになります.

正負0を判断するプログラムへ

```
          LDA   8100H
          ADI   0
          JZ    ZERO
          JM    MINUS
          STA   8101H
          HLT
MINUS:    STA   8102H
          HLT
 ZERO:    MVI   A, FCH
          STA   8103H
          HLT
```

このプログラムの簡単なフローチャートを示すと図6.4のようになります.

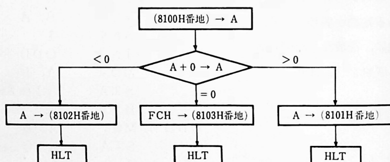


図6.4　データの値が正，0，負によって3方向に分岐する例

4. 繰り返し処理

前節で述べたプログラムの実行の流れを変える命令を用いますと，同じ一連の命令を繰り返し実行するプログラムを作ることができます.

例6－8　1から10までを加算して結果を8100H番地のメモリへ記憶するプログラムは右のようになります.

```
          MVI   A, 0
          MVI   B, 0
          MVI   C, 10
LOOP:     INR   B
          ADD   B
          DCR   C
          JNZ   LOOP
          STA   8100H
          HLT
```

このプログラムはカウンタとしてCレジスタを使っています．すなわちCレジスタに10を入れておき加算するごとに１を引いていきます．そして，Cレジスタの値が０になるまで第４行目から第６行目までの演算を繰り返し行っています．

例６－９　例６－８のプログラムで１から10までの加算を行っている途中で和が30をこえれば，それ以上加算を行わずに，その結果を8100H番地のメモリに記憶し，もし30をこえなければ加算結果を8101H番地のメモリに記憶するプログラムは右のようになります．

```
       MVI   A, 0
       MVI   B, 0
       MVI   C, 10
LOOP:  CPI   30
       JP    OVER
       INR   B
       ADD   B
       DCR   C
       JNZ   LOOP
       STA   8101H
       HLT
OVER:  STA   8100H
       HLT
```

このプログラムは第４行目のCPI命令でＡレジスタに入っている和と30とを比較して，和が30をこえると第５行目のJP命令によって繰り返し演算からはずれてラベルがOVERの文まで分岐します．

例６－10　10×３を計算して，結果を8100H番地のメモリに記憶するプログラムは右のようになります．

```
       MVI   A, 10
       MVI   B, 2
LOOP:  ADI   10
       DCR   B
       JNZ   LOOP
       STA   8100H
       HLT
```

このプログラムは第３行目から第５行目までで10に10を２回加算しています．

例６－11　30÷３の商を計算して，結果を8100H番地のメモリへ記憶するプログラムは右のようになります．

```
       MVI   A, 30
       MVI   B, 0
LOOP:  INR   B
       SUI   3
       JNZ   LOOP
       MOV   A, B
       STA   8100H
       HLT
```

このプログラムは第３行目から第６行目までで，結果が０になるまで30から３を順次減算し，減算した回数をＢレジスタで数えています．

例6－12　定数38を2進数で表したときに1となるビットの数を求めて，それをAレジスタに入れるプログラムは右のようになります．

```
        MVI   A, 38
        MVI   B, 8
        MVI   C, 0
LOOP :  RAL
        JNC   JUMP
        INR   C
JUMP :  DCR   B
        JNZ   LOOP
        MOV   A, C
        HLT
```

このプログラムは，RAL 命令を用いて1回左へ桁移動するごとに第5行目の JNC 命令で，CY フラグが1なら第6行目の INR 命令でCレジスタに1を加え，CY フラグが0ならラベルが JUMP の文まで分岐しています．このことを8回繰り返しますと定数38の2進数表示において1となるビットの数が求まります．

5. データの入出力

マイクロコンピュータの CPU にインターフェスを介して入出力装置を接続しますと，それを通じてデータの入出力を行うことができます．入出力用のインターフェイスは CPU との間にデータ，制御信号，番地を転送するための共通バスをもち，そのほかにいくつかの入出力装置を接続する端子があります．たとえば，8255と名付けられた入出力インターフェイス用 LSI を用いて入出力装置を接続した様子が図6.5に示されています．8255の場合は3台までの8ビットの入出力装置A，B，Cを接続することが可能です．なお，端子Cは上位の4ビットと下位の4ビットを独立に用いることもできます．以下では8255を用いる場合の入出力についで述べます．

三つの入出力装置を用いる前に，まずどの装置をどのように用いるかを指定しておく必要があ

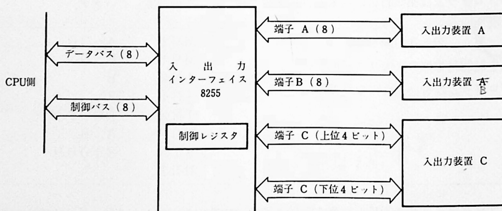


図6.5　入出力インターフェイス用 LSI 8255 と CPU との接続．( )内はビット数を示す．

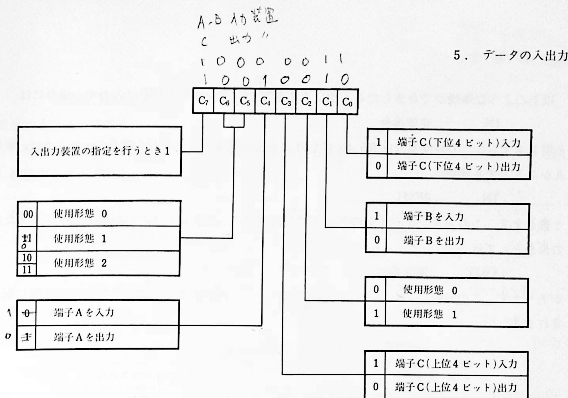


図6.6 8255による入出力装置の指定信号と制御レジスタ

表6.3 8255による入出力装置の使用形態

使用形態	意　　味
0	三つの装置を入力または出力装置として使う.
1	装置A, Bを入力または出力装置として, 装置Cを制御用に使う.
2	装置Aを入力と出力装置に使う.

ります. たとえば装置Aを入力装置として装置Bを出力装置として用いるというような指示を与えておく必要があります. 入出力装置の指示はインターフェイス用LSI内にある8ビットの制御レジスタCRを用い, この各ビットに図6.6に示した入出力装置の指定信号を与えることによって行います. ここで入出力装置の使用形態としては表6.3に示すように3通りあります. たとえば装置A, Bを入力装置とし, Cを出力装置として用いる場合は使用形態は0であり, 制御レジスタを

10010010_2 (92H)

とすればよいのです.

制御レジスタに指定信号を与えるには, プログラムの最初にまず, その信号をAレジスタに入れてから, それをCPUからインターフェイスに出力する命令OUTを用います. このとき, 制御レジスタの装置番号は0FBHですから, これをOUT命令の処理条件に書きます. たとえば, 装置A, Bを入力装置とし, Cを出力装置として用いる場合は, プログラムの最初に

```
MVI     A, 92H
OUT     0FBH
```

と書いておきます.

以上のような準備ができましたら，次にある装置からデータを CPU に入力する場合には

IN　　　装置番号

と書きます．ここで装置番号は表6.4に示したように各装置に付けられた番号で，たとえば，装置Aから入力する場合は

IN　　　0F8H

と書きます．このとき，1バイトのデータが入力装置AからAレジスタに転送されます．また出力命令としては

OUT　　　装置番号

があります．この命令によってAレジスタにあるデータが装置番号で指定された出力装置に転送されます．

表6.4　8255の装置番号

装置名	A	B	C	CR(OUTのときのみ)
装置番号	0F8H	0F9H	0FAH	0FBH

一方，外部の装置に1ビットの制御信号を送るときには端子Cを用いることができます．そのときまず端子Cを出力装置として指定してから，端子Cのどのビットに1または0の信号を送るかを指定するために図6.7の指定コードの値を表6.4に示してある装置番号を用いて CR へ送ります．たとえば端子Cのビット7の値を1にしたいときには

MVI　　　A, 0FH

OUT　　　0FBH

と書いて，制御レジスタに

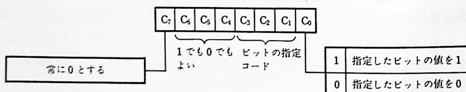


図6.7　端子Cの各ビットに1または0を指定する制御信号

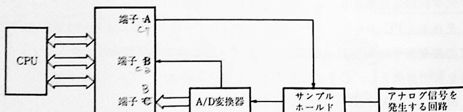


図6.8　アナログ信号を A/D 変換してディジタルデータを CPU に転送する例

00001111₂ (0FH)

を与えるとよいのです．

外部機器を制御する例として，アナログ信号をA/D変換してディジタル信号のデータを CPU に転送する場合の構成が図6.8に示されています．

例6－13　アナログ信号をディジタル量に変換して8100H番地以降のメモリに記憶するプログラムは右のようになります．ただしデータの個数は256以下とします．

```
        MVI   A, 83H
        OUT   0FBH
        LXI   H, 8100H
ADC1:   MVI   A, 0FH     } 端子Cの
        OUT   0FBH       } ビット7を1
        MVI   A, 0EH     }
        OUT   0FBH       } PC7 =0
ADC2:   IN    0FAH
        ANI   08H
        JNZ   ADC2
        IN    0F9H
        MOV   M, A
        MOV   A, L
        CPI   0FFH
        JZ    ADC3
        INX   H
        JMP   ADC1
ADC3:   HLT
```

第1行目と第2行目で使用形態の0を選択し，端子Bを入力，端子Cの上位4ビットを出力，下位4ビットを入力として用いるように選択するために10000011₂ (83H) をいったんAレジスタに入れてから CR に入れています．このとき端子Aは任意でかまいません．第4行目から第7行目まで端子Cのビット7からサンプリングパルスをサンプルホールド回路へ送っています．A/D変換器で変換が終了すると，A/D変換器から端子Cのビット3へ信号が入るように回路を作っておきますと，第8行目から第10行目まで端子Cのビット3の信号が入ってくるのを待ってから，端子Bのデータを取り込みます．

7章　サブルーチンの考え方

1.　はじめに

プログラムの流れの中で，まとまった機能を持つ部分があったり，何度も用いる所がある場合には，その部分をプログラムの流れからとりだして独立させておいて，これを引用する形でプログラムを書くと，全体のプログラムの流れが整理されます．このとき独立させた部分をサブルーチンといい，最初に実行する文のある主となるプログラムを主プログラム（メインプログラム）といいます．この章ではサブルーチンの定義とそれを引用する手法について述べることにします．

サブルーチンの効用はいろいろあります．たとえば，プログラムの中で四則演算のプログラムはコンピュータに計算をさせる場合に大変多く使われます．もし，四則演算をするたびに，この四則演算プログラムを繰り返し使用していたのではプログラムが大変長くなってきます．そこで四則演算プログラムをひとつの独立したプログラムとしておいて，必要に応じて呼びだす，つまりサブルーチンプログラムとしておけばプログラム全体の長さも短かくなります．

ほかにも，サブルーチンを利用するとプログラムを作り易くなるという利点があります．つまり，プログラムを作る場合に，おおまかにプログラムを考えておきます．それからプログラムの細かい部分はサブルーチンプログラムとして作っていくのです．このようにすると，プログラムが構造化されプログラムを作るのが大変楽になります．

このように有用なサブルーチンもそれを利用する場合にいくつか知っておかなければならないことがあります．次節からそのことについて述べていくことにします．

2.　サブルーチンの定義

2.1　無条件リターンのサブルーチン

サブルーチンには主プログラムからそれを引用した場合，その実行が終われば無条件に主プログラムに戻る無条件リターンのサブルーチンと，サブルーチンで処理した結果によって主プログラムに戻る条件付きリターンのサブルーチンとがあります．無条件リターンのサブルーチンは，その先頭の文にラベル(標識)を付け，最後の文には，主プログラムに戻るための RET (return) 命令を書きます．

```
SUB : ―――
        ⋮
      RET
```

例7－1　Aレジスタにある定数を２回左へ桁移動して，そのあとは最下位桁から順に０がはいるプログラムをサブルーチンにすると右のようになります．

```
SUB:  RLC
      ANI   FEH
      RLC
      ANI   FEH
      RET
```

このサブルーチンは，RLC 命令で１ビット左へ環状に桁移動して，ANI 命令で最下位桁を０にしています．そして最後の RET 命令で実行の流れは，無条件に主にプログラムに戻ります．

2．2　条件付きリターンのサブルーチン

前の例のようにサブルーチンの最後に RET 命令があれば，実行の流れは無条件に主プログラムに戻ります．ところが，サブルーチン内で処理した結果によって，主プログラムに戻るかどうかを決定したいことがあります．このときに使われるのが表７．１に示した条件付きリターン命令です．これらの命令はフラグの値によって主プログラムに戻るかどうか決定します．戻らない場合はその命令を無視して次の命令を実行します．

たとえばサブルーチン内の演算の結果，Ｚフラグが０になったときに主プログラムに戻るためにはサブルーチン内に RNZ 命令を含み

```
SUB: ―――
        ⋮
       RNZ
        ⋮
       RET
```

の形に書きます．このようにすると，RNZ 命令を実行するときにＺフラグが０なら主プログラムに戻りますが，１のときは RNZ 命令の次の命令を順次実行して最後の RET 命令によって主プログラムに戻ります．

例7－2　8100H番地のメモリに記憶してある定数の絶対値をとってAレジスタへ転送するサブルーチンは右のようになります．

```
SUB:  LDA   8100M
      ADI   0
      RP
      MOV   B, A
      MVI   A, 0
      SUB   B
      RET
```

Aレジスタに8100H番地のメモリの定数を転送して，それが正なら第３行目の条件付きリターン命令によって主プログラムに戻り，負なら第４行目から第６行目までを正になおしてから主プログラムに戻ってきます．

表７．１　フラグの値によって主プログラムに戻る条件付きリターン命令

フラグ／フラグの値	Z	S	CY	P
0	RNZ (return if not zero)	RP (return if positive)	RNC (return if no carry)	RPO (return if parity odd)
1	RZ (return if zero)	RM (return if minus)	RC (return if carry)	RPE (return if parity even)

例７－３　０以外の二つの１バイトの定数の掛算を行うサブルーチンMLTYは右のようになります．ただし二つの定数はAレジスタとBレジスタにあらかじめ入っているものとして，また答は２バイトの定数でHLレジスタ対に入れるものとします．

```
MLTY:  MOV   E, B
       MVI   D, 0
       MOV   L, D
       MOV   H, D
LOOP:  DAD   D
       DCR   A
       JNZ   LOOP
       RET
```

このサブルーチンは，ａをｂ回加算してａ×ｂの掛算を行っています．Aレジスタをカウンタとして利用して，最初そこにｂを入れておき第５行目から第７行目までの文によってカウンタの値が０になるまでHLレジスタ対にａを加算しています．

例７－４　正の定数ａ，ｂ（ただしａ≧ｂ）の割算ａ÷ｂを行うサブルーチンDIVは右のようになります．ただし，ａはAレジスタに，ｂはBレジスタに入っているものとします．また商はCレジスタに，余りはDレジスタに入れることにします．

```
DIV:   MVI   C, 0
LOOP:  INR   C
       SUB   B
       JZ    JUMP
       CMP   B
       JP    LOOP
JUMP:  MOV   D, A
       RET
```

このサブルーチンは，Aレジスタの値からBレジスタの値を，Aレジスタの値が０かBレジスタの値より小さくなるまで引いていきます．引いた回数をCレジスタに入れておき，それを商とし，Aレジスタに残った数を余りとしています．この例のフローチャートは図７．１に示されています．

条件付きリターン文を用いるサブルーチンにおいて，たとえば

```
SUB:        ⋮
A           RNZ
            ⋮
            JMP    A
```

のように繰り返し処理を行う場合は，サブルーチンの最後に RET 命令を含まないこともあります．

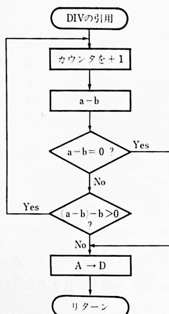


図7.1 例7-4のフローチャート

例7－5 1からCレジスタにある整数までを加算するサブルーチンは右のようになります．

```
SUB:    MVI    A, 0
        MVI    B, 0
LOOP:   INR    B
        ADD    B
        DCR    C
        RZ
        JMP    LOOP
```

3．サブルーチンの引用

3．1 無条件引用

主プログラムにおいて，サブルーチンを引用するときには，CALL (call) 命令の処理条件にサブルーチンの先頭の文のラベル，たとえば SUB を用いて

```
CALL    SUB
```

と書きます．なお，サブルーチンは主プログラムの前または後のどちらに書いてもかまいませんが，本書では主プログラムの後に書くことにします．

例７－６　主プログラムのA，Bレジスタに保持している定数２，３をサブルーチンで加算してから左へ桁移動して，その結果を主プログラムによって8200H番地のメモリに記憶するプログラムは右のようになります．

```
MAIN:  MVI   A, 2
       MVI   B, 3
       CALL  SUB
       LXI   H, 8200H
       MOV   M, A
       HLT
*
SUB:   ADD   B
       RLC
       RET
```

例７－７　例７－３を用いて３×２を計算し，答えをAレジスタに入れるプログラムは右のようになります．

```
MAIN:  MVI   A, 3
       MVI   B, 2
       CALL  MLTY
       MOV   A, L
       HLT
*
MLTY:  MOV   E, B
       MVI   D, 0
       MOV   L, D
       MOV   H, D
LOOP:  DAD   D
       DCR   A
       JNZ   LOOP
       RET
```

例７－８　8200H番地のメモリに記憶してある１バイトの２進数を，BCDに変換するプログラムは右のようになります．ただし，例７－４の割算サブルーチンDIVを用います．

```
       STA   8200H
       CPI   100
       JM    JP1
       MVI   B, 100
       CALL  DIV
       MOV   H, C
       MOV   A, D
JP1:   CPI   10
       JM    JP2
       MVI   B, 10
       CALL  DIV
       MOV   A, C
       RLC
       RLC
       RLC
       RLC
       ADD   D
       MOV   L, A
```

```
                    JMP     JP3
        JP2:        MOV     L, A
        JP3:        SHLD    8100H
                    HLT
```

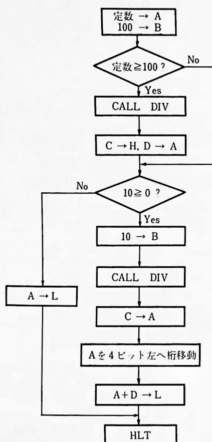


図7.2　例4-8のフローチャート

このプログラムではフローチャートに示した手順に従って2進数の10000000_2をBCDに変換しています．まず第6行目までで10000000_2を100で割り，商をHレジスタの下位4ビットに入れています．第16行目までで余りを10で割って，商をLレジスタの上位4ビットに入れています．第17行目のADD命令でその余りをLレジスタの下位4ビットに入れます．したがってBCDはHLレジスタ対に入り，SHLD命令でBCDは8100Hと8101H番地のメモリに記憶されます．

3．2　条件付き引用

主プログラム中の処理結果，すなわちフラグの値によってサブルーチンを引用したいことがあります．このときに用いられる条件付き引用命令が表7．2に示されています．たとえばZフラグが0のとき，ラベルがSUBのサブルーチンを引用するためには

```
CNZ    SUB
```

と書きます．この命令は，Zフラグが1のときには無視されて，この次の文を実行します．この

ようにCNZ，CNC，CPO，CP命令はそれぞれに該当するフラグの値が0のときにのみサブルーチンを引用しますが，フラグの値が1のときは無視されます．同様にCZ，CC，CPE，CM命令はフラグの値が1のときにのみサブルーチンを引用しますが，0のときは無視されます．

例7－9　Aレジスタに入れた定数が，偶数ならサブルーチンでその値と定数3とを加算してAレジスタへ入れ，奇数ならそのままにしておくプログラムは右のようになります．

```
       MVI   A，定数
       MOV   B，A
       ANI   1
       CZ    SUB
       HLT
SUB：  MVI   A，3
       ADD   B
       RET
```

表7.2　フラグの値によって用いるサブルーチンの条件付き引用命令

フラグ ／ フラグの値	Z	S	CY	P
0	CNZ (call if not zero)	CP (call if positive)	CNC (call if no carry)	CPO (call if parity odd)
1	CZ (call if zero)	CM (call if minus)	CC (call if carry)	CPE (call if parity even)

例7－10　8100H番地と8101H番地のメモリに記憶されている定数の和が，偶数なら1からその数までの和を求めて，奇数なら1を8102H番地のメモリに記憶するプログラムは右のようになります．

```
MAIN：  LDA   8100H
        MOV   B，A
        LDA   8101H
        ADD   B
        MOV   C，A
        ANI   1
        CZ    SUB
        STA   8102H
        HLT
*
SUB：   MVI   A，0
        MVI   B，0
LOOP：  INR   B
        ADD   B
        DCR   C
        RZ
        JMP   LOOP
```

このプログラムでは主プログラムの第4行目で加算を行い，その結果を第6行目と第7行目で調べて偶数ならSUBを引用し，奇数ならそのまま1を8102H番地のメモリに記憶するサブルーチンは例7-5のSUBと同じです．

4. 二つ以上のサブルーチンの引用

一つのプログラムの中にサブルーチンをいくつ書いてもよく，それらは必要に応じて何回でも使用することができます．またサブルーチンから他のサブルーチンを引用することもできます．

例7－11　Aレジスタにある定数が正または0ならサブルーチン1を引用して定数3を加算し，負ならそのままそれを主プログラムで8100H番地のメモリに記憶した後，再びサブルーチン2で左へ桁移動してその結果を8101H番地のメモリに記憶するプログラムは右のようになります．

```
MAIN:  MVI    A, 定数
       CPI    0
       CP     SUB1
       STA    8100H
       CALL   SUB2
       STA    8101H
       HLT
SUB1:  MVI    B, 3
       ADD    B
       RET
SUB2:  RLC
       RET
```

このプログラムの流れは図7．3に示したように条件によってサブルーチンを引用し，後でまたサブルーチン2を引用しています．

例7－12　例7－9で加算結果が6以上のときは，サブルーチン2で定数4を加算するプログラムは右のようになります．

```
MAIN:  MVI    A, 定数
       MOV    B, A
       ANI    1
       CZ     SUB1
       HLT
SUB1:  MVI    A, 3
       ADD    B
       CPI    6
       CP     SUB2
       RET
SUB2:  ADI    4
       RET
```

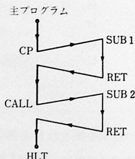


図7.3　サブルーチンを2回引用する例

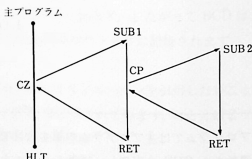


図7.4　サブルーチンから他のサブルーチンを引用する例

このプログラムは図7．4に示すように主プログラムから条件によってサブルーチン1を引用し，さらに条件によってそこから再びサブルーチン2を引用しています．

5． スタックの利用

5．1 サブルーチン実行後の戻り番地の記憶

サブルーチンを引用してその処理を終えれば，実行の流れはそれを引用したもとのプログラムの引用文の次に戻ります．そのため戻ったときに次に実行する文の番地を記憶しておく必要があります．この番地はサブルーチンを引用したときに，スタックと呼ばれるメモリ領域の2バイトを用いて自動的に記憶され，サブルーチン処理をすべて終えて，もとのプログラムに戻ったときに自動的にプログラムカウンタに転送されます．

スタックの番地はSP（スタックポインタ）と呼ばれる2バイトのレジスタを用いて指定します．SPはプログラムで指定されない限り，最初は特定の番地を示しています．たとえばTK-85の場合は初期値として8391H番地を保持しています．サブルーチンを引用する命令を実行すると，そのサブルーチンの処理を終了したときに戻るべき番地がSPにある番地のメモリの手前2バイトに記憶されます．このときSPの数値から2を引いた番地のメモリに下位1バイトが，1を引いた番地のメモリに上位の1バイトが入り，さらにSPの値は2が引かれて，次のスタックが利用できる状態になります．

またサブルーチンの引用が終わって実行がもとに戻ると，SPの値は2が加えられます．

> **例7－13**　例7－6のプログラムを実行するとき，スタックとその番地を示すＳＰはどのようになるでしょうか．

例7-6のプログラムは次のように8000H番地から始まるメモリに記憶してから実行するものとします．

```
番地
8000H    MAIN:  MVI    A, 2
8002H           MVI    B, 3
8004H           CALL   SUB
8007H           LXI    H, 8200H
800AH           MOV    M, A
800BH           HLT
8100H    SUB:   ADD    B
8101H           RLC
8102H           RET
```

SPはあらかじめ指定しないかぎり，8391Hを保持しています．第3行目のCALL命令でサブルーチンSUBを引用するときには，その処理が終わって主プログラムに戻ったときに，次に実行される8007H番地が，スタックの838FH～8390H番地に図7．5のような順で記憶されます．



図7.5 サブルーチンを引用するときのスタックと SP の様子

このとき SP の値は2引かれて 838FH となります．サブルーチンの引用が終わると RET 命令により，スタックに記憶されている戻り番地がプログラムカウンタに転送され，SP の値は8391H に戻ります．

スタックの内容を知るために用いる命令としては，XTHL (exchange H and L with top of stack) 命令

XTHL

があり，この命令は図7．6のように SP で指定される番地のメモリの内容をLレジスタに，その次の番地のメモリの内容をHレジスタに転送します．このとき SP の値は変化しません．これと同様のことは POP 命令を用いて

POP　　H

と書いても実行できますが，このときは SP の値は2加えられます．

例7－14　例7－12を実行したときスタックにどのようなデータが入っていくかを調べるプログラムは下のようになります．

番地			
8000H	MAIN:	MVI	A, 定数
8002H		MOV	B, A
8003H		ANI	1
8005H		CZ	SUB1
8008H		HLT	
8009H	SUB1:	XTHL	
800AH		SHLD	8100H
800DH		MVI	A, 3
800FH		ADD	B
8010H		CMI	6
8012H		CP	SUB2
8015H		RET	
8016H	SUB2:	XTHL	
8017H		SHLD	8102H
801AH		ADI	4
801CH		RET	

このプログラムでは二つのサブルーチンを用いた場合に，スタックがどのように用いられるかを調べています．まず最初，SP は8391H番地を示していますが主プログラムで SUB1 を引用したときには，戻り番地である8008H番地が図７．７(a)のように記憶されます．このことを確かめるために，8009H～800AH番地の命令で，8008Hを図７．７(c)に示したように8100H～8101H番地のメモリに記憶しておきます．

また SUB1 において，SUB2を引用したときには戻り番地である8015H番地は図７．７(b)のように記憶されます．このときも同様に8016H～8017H番地の命令で8015Hを8102H～8103H番地のメモリに記憶しておきます．主プログラムに戻るときには，逆に 838DH 番地から8391H番地へと順次下へ戻ります．最後に8100H番地から8103H番地のメモリの内容を調べると SUB1, SUB2 を引用したときにスタックに入った値が分かります．

SPの初期値をプログラマが指定したい場合，たとえば8300H番地としたい場合，6章で述べましたように

LXI　　SP, 8300H

とすればよいのです．また HL レジスタ対に8300H番地を入れておいてから SPHL (move H and L to stack pointer) 命令

SPHL

を書いても，HL レジスタ対の内容が SP に転送されます．

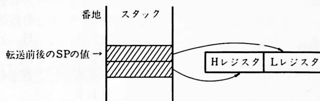


図7.6　XTHL 命令によるスタックの内容の読み出し

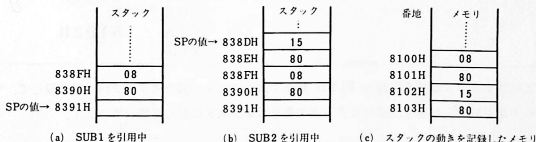


図7.7　例7-12の処理中の SP とスタックの動き

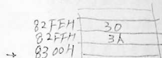
例７－15　ＳＰの値を8300Hに指定してスタックを用いたいときは右のようにします．	ＬＸＩ　　Ｈ，８３００Ｈ ＳＰＨＬ ＨＬＴ

5.2 レジスタ対のデータの退避

8085 で用意されているレジスタ対は AF, BC, DE, HL の五つですが，もっと多くのレジスタ対を用いたいときや，またはサブルーチンを引用する前のレジスタ対のデータをサブルーチン実行中に一時保護しておきたいときにもスタックが利用できます．たとえば，HL レジスタの値をスタックへ退避するには

PUSH　　H

と書くとよいのです．



例7－16　SPの値を8300H番地に設定してから，HLレジスタ対に入っている定数3A30Hをスタックに退避させるプログラムは右のようになります．	LXI　SP，8300H LXI　H，3A30H PUSH　H HLT

このプログラムでは 82FFH 番地に 3AH が，82FEH 番地に30Hが入っています．

例7－17　8100H～8101H番地にある定数10と5の加算を行いサブルーチンで減算を行った結果と加えて，8103H番地に記憶するプログラムは右のようになります．	

```
        LDA    8100H
        MOV    B，A
        LDA    8101H
        ADD    B
        PUSH   PSW
        CALL   SUB
        POP    PSW
        LXI    H，8102H
        ADD    M
        STA    8103H
        HLT
SUB：   LDA    8101H
        MOV    B，A
        LDA    8100H
        SUB    B
        STA    8102H
        RET
```

このプログラムは第6行目の PUSH 命令でAレジスタの値をスタックへ一時退避して，サブルーチンを引用した後で，またスタックの値をAレジスタに戻して用いています．

6. 割り込み処理

いままでに述べたサブルーチンは，あらかじめ，決められたときに引用してプログラムの流れを変更しましたが，あるプログラムを実行中に周辺機器から信号を加えることによってプログラムの流れを，別の処理を行うサブルーチンに変更することができます．この処理を割り込み処

理といい，これを行う命令を割り込み命令といいます．この処理は周辺機器とのデータの転送などに用いられます．

主プログラムにおいて割り込まれてもよいときは EI (enable interrupt) 命令

「EI」

を書き，割り込まれては困る場合は DI (disable interrupt) 命令

「DI」

を書いておきます．

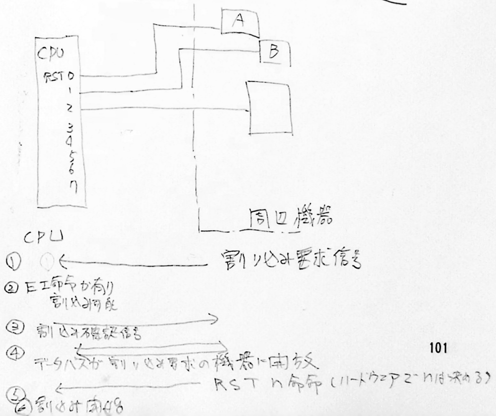
あるプログラムを実行中に割り込みが発生するまでの経過は次のようになります．まず割り込みたい周辺機器から CPU へ割り込み要求信号が送られてきます．そのとき EI 命令があって CPU が割り込み可能ならば，現在実行中の命令の処理が終わった直後に割り込みができる状態になります．次に CPU から周辺機器へ割り込み確認信号が送られ，データバスが割り込みを要求した機器に対して開放されます．このとき，サブルーチンのときと同様に，戻り番地は自動的にスタックに記憶されます．次に周辺機器から，割り込み実行命令である RST (restart) 命令

「RST　n」

がデータバスに乗せられて CPU に入ってきます．この命令は各周辺機器にそれぞれ別の処理条件をもつようにハードウェアで用意する命令であり，プログラム中にはでてきません．処理条件の n は 0 から 7 までの 8 種類までの整数がとれます．

RST 命令は，無条件分岐命令と同じ機能をもちます．すなわち，割り込みをする周辺機器からその機器に割り当てられた RST 命令がくると，プログラムの流れをいったん分岐します．

割り込みが発生しているときには他の割り込みは受け付けられず，複数の割り込みが同時に入ってきたときは，RST 命令の処理条件の小さい数を用いるほど優先順位が高くなります．



8章　モニタの詳細な説明

1．はじめに

どんなコンピュータもプログラムなしでは，何も仕事をすることができません．これらの処理プログラムはコンピュータのメモリ内部に書かれ，実行されます．したがって，これらの処理プログラムをメモリに対してアクセスするような基本的な機能やデバッグ機能等が必要になります．

モニタプログラムは，一般的に処理プログラムに対して制御プログラムと呼ばれ，各処理プログラムを効率良く管理し，コンピュータの中枢機能となるプログラムです．大型計算機にはOSと呼ばれる巨大な管理プログラムが存在しますが，モニタはこのOSのマイクロコンピュータ版だと思ってください．

TK-85のモニタは2kバイトの大きさで構成されています．全体の構成を理解していただくために2節以降にフローチャートおよび各部分の詳細を説明します．3章に述べた，操作方法を頭に浮かべながら一つ一つ理解していって下さい．

2．フローチャート

モニタ全体の構成を把握していただくために図8.1にフローチャートを示します．

リスティングを解説する際に全体の構成が分かっていないと，部分的な解読ができても，それぞれのつながりがうまくいかず，結局断片的な理解で終ってしまいます．したがっていきなりリスティングの解読を行なわずに，まずこのフローチャートによって全体を把握してください．

〈フローチャートの用語説明〉

(AR)：アドレスレジスタ――2バイト

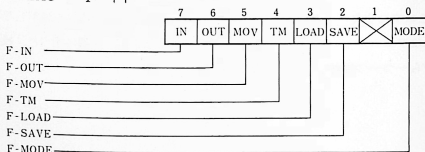
(DR)：データレジスタ――2バイト

(PC)：プログラムカウンタセーブエリア――2バイト

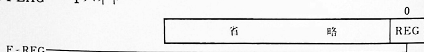
(BD)：ブレイクデプス――2バイト

(BP)：ブレイクポイント――2バイト

MODE FLAG――1バイト



REG FLAG――1バイト

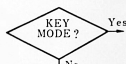




○F-MOVのビットテスト
ON：“1”
OFF：“0”



○MODE FLAG 8ビットすべてのビットテスト
ON：NON 0
OFF：0



○キー入力データの判別

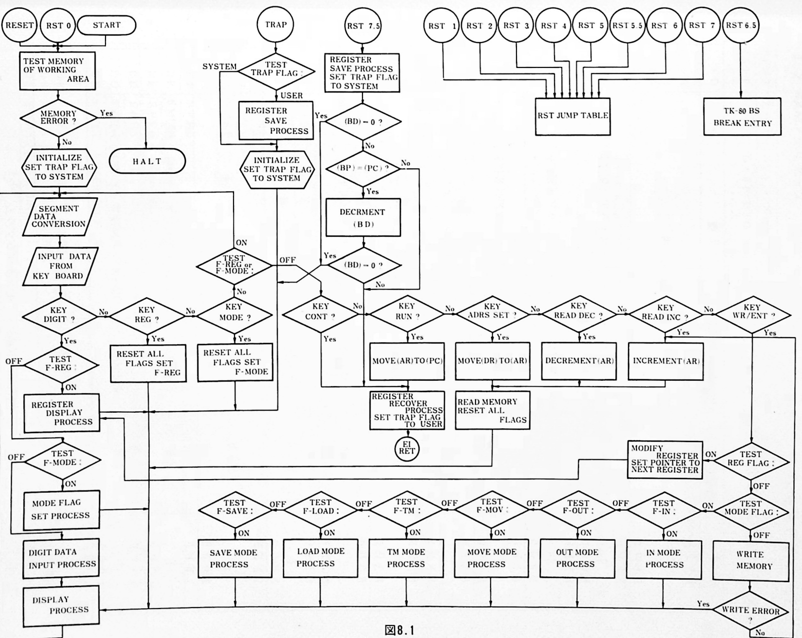

RESET
RST 0
START
TEST MEMORY OF WORKING AREA
MEMORY ERROR ?
HALT
INITIALIZE SET TRAP FLAG TO SYSTEM
SEGMENT DATA CONVERSION
INPUT DATA FROM KEY BOARD
KEY DIGIT ?
TEST F-REG ?
REGISTER DISPLAY PROCESS
TEST F-MODE ?
MODE FLAG SET PROCESS
DIGIT DATA INPUT PROCESS
DISPLAY PROCESS
KEY REG ?
RESET ALL FLAGS SET F-REG
KEY MODE ?
RESET ALL FLAGS SET F-MODE
TEST F-REG or F-MODE ?
TRAP
TEST TRAP FLAG ?
SYSTEM
USER
REGISTER SAVE PROCESS
KEY CONT ?
RST 7.5
REGISTER SAVE PROCESS SET TRAP FLAG TO SYSTEM
(BD) = 0 ?
(BP) = (PC) ?
DECRMENT (BD)
KEY RUN ?
MOVE(AR)TO(PC)
REGISTER RECOVER PROCESS SET TRAP FLAG TO USER
EI RET
KEY ADRS SET ?
MOVE(DR) TO(AR)
READ MEMORY RESET ALL FLAGS
KEY READ DEC ?
DECREMENT(AR)
KEY READ INC ?
INCREMENT(AR)
KEY WR/ENT ?
TEST REG FLAG ?
MODIFY REGISTER SET POINTER TO NEXT REGISTER
TEST MODE FLAG ?
WRITE MEMORY
WRITE ERROR ?
TEST F-SAVE ?
SAVE MODE PROCESS
TEST F-LOAD ?
LOAD MODE PROCESS
TEST F-TM ?
TM MODE PROCESS
TEST F-MOV ?
MOVE MODE PROCESS
TEST F-OUT ?
OUT MODE PROCESS
TEST F-IN ?
IN MODE PROCESS
RST 1
RST 2
RST 3
RST 4
RST 5
RST 5.5
RST 6
RST 7
RST 6.5
RST JUMP TABLE
TK-80 BS BREAK ENTRY
Yes
No
ON
OFF

図8.1

3．メモリマップ

3.1　システムメモリマップ

アドレス	容量(バイト)	ROM or RAM	備　　考
FFFF ↑ 9000	28k	－	未使用
8FFF ↑ 8C00	1k	RAM	ユーザーズエリア（要拡張）
8BFF ↑ 8800	1k	RAM	ユーザーズエリア（要拡張）
87FF ↑ 8400	1k	RAM	ユーザーズエリア（要拡張）
83FF ↑ 83B1	55	RAM	モニタワーキングエリア
83C8 ↑ 83B1	24	RAM	RSTジャンプテーブル
83B0 ↑ 8391	32	RAM	モニタスタックエリア
8390 ↑ 8000	913	RAM	ユーザーエリア
7FFF ↑ 2000	24k	－	未使用
1FFF ↑ 1800	2k	PROM	ユーザーズエリア（オプション）
17FF ↑ 1000	2k	PROM	ユーザーズエリア（オプション）
0FFF ↑ 0800	2k	PROM	ユーザーズエリア（オプション）
07FF ↑ 0000	2k	MASK ROM	モニタプログラム

3．2 RAMメモリマップ

アドレス	ラベル	シンボル	名　称	名　考
83FF		DISP DIG 1	セグメントデータバッファ	DMA転送によって，常時このエリアのセグメントデータが，LED上に表示されている
FE		〃 2		
FD		〃 3		
FC		〃 4		
FB		〃 5		
FA		〃 6		
F9		〃 7		
F8	DIG	〃 8		
F7		DISP WORD 4	ディスプレイレジスタ	LEDディスプレイに表示するための16進データをセットするレジスタ
F6		〃 3		
F5		〃 2		
F4	DISP	〃 1		
F3		〔H〕	ブレイクデプス	ブレイク動作時に，ループ回数をセットするレジスタ
F2	SVBD	BREAK DEPTH〔L〕		
F1		〔H〕	ブレイクポイント	ブレイク番地をセットするレジスタ
F0	SVBP	BREAK POINT〔L〕		
EF		〔H〕	アドレスレジスタ	モニタがメモリに対して処理を行うときに参照するレジスタ
EE	ADRES	ADRESS REG〔L〕		
ED		〔H〕	データレジスタ	ディジットキーより入力されたデータおよびメモリよりリードされたデータがセットされるレジスタ
EC	DATA	DATA REG〔L〕		
EB		A	CPUレジスタセーブエリア	割り込みがかかった時点における，CPUレジスタの内容を退避するエリア
EA	SVAF	F		
E9		B		
E8	SVBC	C		
E7		D		
E6	SVDE	E		
E5		H		
E4	SVHL	L		
E3		〔H〕		
E2	SVSP	SP〔L〕		
E1		〔H〕		
E0	SVPC	PC〔L〕		
DF	FMODE	MODE FLAG	フラグエリア	モニタ処理時の状態を示すフラグのエリア
DE	FREG	REGISTER FLAG		
DD	FKEY	KEY FLAG		
DC	FTRAP	TRAP FLAG		
DB	ENTCT	ENTER COUNTER		
DA		〔H〕	データセーブエリア	モニタ処理における，一時的なデータ退避エリア
D9		〔L〕		
D8		〔H〕		
D7	SVDAT	SAVE DATA〔L〕		

アドレス	ラベル	シンボル	名称	備考
83D6 D5 D4 D3	ADW1 ADW2 ADW3 ADW4	ADDRESS DISP WORD 1 〃 2 〃 3 〃 4	アドレスディスプレイレジスタ	アドレスディスプレイに表示するディスプレイレジスタデータをセットするレジスタ
D2 D1 D0 CF	DDW1 DDW2 DDW3 DDW4	DATA DISP WORD 1 〃 2 〃 3 〃 4	データディスプレイレジスタ	データディスプレイに表示するディスプレイレジスタデータをセットするレジスタ
CE CD CC	PORTO GOOUT	C 9 OUT PORT ADDRESS D 3	OUT命令エリア	OUT命令実行用エリア
CB CA C9	PORTI GOIN	C 9 IN PORT ADDRESS D B	IN命令エリア	IN命令実行用エリア
C8 C7 C6	RST7	RST7	RST7ジャンプテーブル	RST命令によりジャンプしてくるエリア。このエリアにユーザーがジャンプ命令をセットすれば，RST命令実行後，設定エリアにジャンプすることができる。
C5 C4 C3	RST6	RST6	RST6ジャンプテーブル	
C2 C1 C0	RST5.5	RST5.5	RST5.5ジャンプテーブル	
BF BE BD	RST5	RST5	RST5ジャンプテーブル	
BC BB BA	RST4	RST4	RST4ジャンプテーブル	
B9 B8 B7	RST3	RST3	RST3ジャンプテーブル	
B6 B5 B4	RST2	RST2	RST2ジャンプテーブル	
B3 B2 B1	RST1	RST1	RST1ジャンプテーブル	
83B0 8391	USESP	MONITORS STACK	モニタスタックエリア	モニタが使用するスタックエリア
8390 8000		USERS STACK ↓	ユーザースタックエリア ユーザーエリア	ユーザーが使用できるRAMエリア．なおユーザープログラムを実行すると，8391番地にスタックポインタが設定される．

4．モニタを理解するために

3章の操作方法のところを読んで，モニタがどのような処理を行っているか理解できたことと思います．操作方法が理解できたということは，いわゆる外から見たモニタを理解したわけです．これではまだモニタ内部がブラックボックスのままで，操作は分かっても実際にどのようなアルゴリズムになっているかは依然未知のままです．このセクション以降はモニタ内部を理解するための説明です．

TK-85 モニタの構成は，通常のシステムプログラムの構成とは多少異なっています．通常のシステムプログラムは，一度各ファンクションに分岐したら，各ファンクションの動作が終了するまで多少の例外を除いて，そのシステムプログラムの先頭には戻りません．

ところが TK-85 のモニタは，入力キー一つに対する処理を行ったら，再びモニタの先頭に戻り，同じキー入力ルーチンでキースキャンを行っています．このようなモニタ形式をとった場合，問題になる点があります．それは一つのキーにいくつかの意味を持たせた場合，押されたキーが一体どの意味で押されたものなのか分からないということです．これが，通常のシステムプログラムのように，各ファンクション内がシーケンシャル構造となっている場合は，各ファンクション内におけるキーの意味付けが確定しているので，このような問題は起こりません．

そこで TK-85 モニタでは，いくつかのフラグを設け，それを参照することによって，入力キーの解釈を行っています．

5．分岐とフラグ

ワーキングエリアの中身を見ると，フラグエリアという領域が5バイト分確保されています．それぞれのフラグはプログラムの分岐用に設けられたものです．

まず各フラグを紹介し，それらのフラグがどのような用途に用いられているか，プログラムの流れと共に説明していきましょう．

5.1 モードフラグ（FMODE：83DF 番地）

このフラグ構成は下図のようになっています．

7	6	5	4	3	2	1	0
IN	OUT	MOV	TM	LOAD	SAVE	未使用	MODE

操作方法の章でも述べましたが，モニタは三つのファンクションに分かれています．すなわち，ノーマルファンクション，モードファンクションそしてレジスタファンクションです．

このモードフラグはモードファンクション内部で用いられるフラグです．

MODE キーが押されると，アドレスディスプレイに4ドットのみを表示し，モードファンクションに入ったことを知らせます．

このとき，モニタはモードフラグの MODE をセットし，レジスタフラグおよびエンターカウンタをクリアします．そしてアドレスおよびデータディスプレイレジスタに BLANK のデータを格納し（アドレスおよびデータレジスタには 0 を格納）セグメントデータ変換を行い，再びキー入力待ちになります．

アドレスおよびデータディスプレイレジスタには BLANK のデータが格納されましたが，アドレスディスプレイ上には四つのドットが表示されています．これは，セグメントデータ変換サブルーチン内部でフラグセンスを行い，モードフラグの MODE ビットが立っていたときには，セグメントデータとドットのデータとの論理和をとっているからです．

[MODE]キーが押され，モードフラグの MODE ビットが立つと，キーボードスイッチのディジットモードファンクションキー（[A/SAVE][B/LOAD][C/TM][D/MOV][E/OUT][F/IN]）のモードファクション側が有効になります．

なお，このフラグビットが立っていると，モニタは[MODE][REG]モードファンクションキーおよびハードウェアスイッチ以外のキー入力を受け付けません．

現在，アドレスディスプレイ上に四つのドットが表示され，モードの選択が要求されています．ここで各モードファンクションキーを入力することにより六つのモードのいずれかを選択することができます．

このとき，モニタは各モードファンクションに対応するフラグビットをセットし，MODE ビットをリセットします．その後セグメントデータ変換を行い，再びキー入力待ちになります．

IN モード，OUT モード以外はディスプレイレジスタへの操作を何も施していませんが，アドレスディスプレイ上には各モードに対応するドットが一つ表示され，どのモード内にいるのか一目で分かるようになっています．これも先程と同ようにドットの表示がすべて，セグメントデータ変換ルーチン内のフラグセンスによって制御されているからです．

この結果，モニタは各モードファンクションキーのモードになり，そのモードが終了するか，あるいはディジットキーおよび[WR/ENT]キーを除いたキーが入力されるまでは，モードフラグの対応ビットが立ち続け，各モード処理内部にいる訳です．

なおモニタが各モード処理内部にいる場合は，[WR/ENT]キーは ENTER キーとして作用します．フラグエリア内部に ENTCT が 1 バイト確保され，各モード処理内部で押される ENTER キーのカウントを行っています．これは各モード処理内部で行われる処理が，入力 ENTER キーひとつひとつによって異なっているためです．

5.2　レジスタフラグ（FREG：83DE番地）

このフラグ構成は下図のようになっています．

7	6	5	4	3	2	1	0
BR.D	BR.P	SP	HL	DE	BC	AF	REG

このレジスタフラグはレジスタファンクション内部で用いられるフラグです．

[REG]キーが押されると，ディスプレイに，REG に対応するドットのみを表示し，レジスタファンクションに入ったことを知らせます．

このときモニタはレジスタフラグの REG をセットし，モードフラグおよびエンターカウンタをクリアします．そしてアドレスおよびデータディスプレイレジスタに BLANKのデータを格納し，（アドレスおよびデータレジスタには 0 を格納），セグメントデータ変換を行い，再びキー入力待ちになります．

アドレスおよびデータディスプレイレジスタには BLANK のデータが格納されましたが，ディスプレイ上には一つのドットが表示されています．これもモードフラグの場合と同様にセグメントデータ変換ルーチン内で，レジスタフラグをセンスしてドット制御を行っているからです．

[REG]キーが押され，レジスタフラグの REG ビットが立つと，キーボードスイッチのディジットレジスタファンクションキー（[0/AF][1/BC][2/DE][3/HL][4/SP][5/BRP][6/BRD]）のレジスタファンクション側が有効になります．

なお，このフラグビットが立っていると，モニタは[MODE][REG]レジスタファンクションキーおよびハードウェアスイッチ以外のキー入力を受け付けません．

現在，ディスプレイ上に一つのドットが表示され，表示レジスタの選択が要求されています．ここで各レジスタファンクションキーを入力することにより，五つのペアレジスタとブレイクポイント，そしてブレイクデプスの表示を選択することができます．

このときモニタは各レジスタに対応するビットをセットし，REG ビットをリセットします．その後選択されたレジスタの名前およびその内容を表示し，再びキー入力待ちになります．

レジスタ表示機能だけならばレジスタフラグの REG ビットのみで十分です．しかし，レジスタファンクションは単にレジスタの内容を表示するだけではなく，その内容を修正したり連続して他のレジスタの内容を表示させる機能を有しています．したがって，現在対象としているレジスタのポインタを設定する必要があります．レジスタフラグの REG を除く各ビットはこのポインタとしての役割を担っているのです．

5.3 トラップフラグ（FTRAP：83DC 番地）

[MON]キーを押すと，TRAP 割り込みがかかります．

TRAP 割り込みは，インターラプトイネーブルや，割り込みマスクに関係なく割り込みがかかります．したがってモニタ内部でも割り込みがかかってしまい，[MON]キーがモニタ内部で押されたのか，ユーザーのアプリケーションプログラム内部で押されたのか見極める必要があります．トラップフラグはこれらの判断の手段として設けられたフラグです．

トラップフラグはモニタ内部では00H（SYSTEM）という値をとり，ユーザープログラム内部では FFH（USER）という値をとります．

0 番地スタート，TRAP 割り込み処理ルーチン突入時および RST 7.5 割り込み処理ルーチン



突入時等，モニタプログラムへの入口でトラップフラグは00Hにセットされます．

またユーザープログラムへのジャンプ時，つまりモニタプログラムの出口でトラップフラグはFFHにセットされます．

なお，トラップフラグにともなう処理の説明は6節を参照して下さい．

5．4　キーフラグ（FKEY：83DD番地）

新たなキー入力を検出するために，キーフラグが設けられています．

キーフラグは，キーボードスイッチが押されている状態ではFFHという値をとり，キーボードスイッチが離されている状態で00Hという値をとります．

なお詳細な説明は10章4節の入力用サブルーチンの項を参照して下さい．

6．割り込み処理とイニシャライズ

8085Aには5本の割り込み入力（INTR，RST 5.5，RST 6.5，RST 7.5そしてTRAP）があります．TK-85では，RST 5.5を除く他の割り込み入力を使用しています．これらの割り込みに対してモニタがどのような処理を行っているのか，また電源投入時あるいはリセットボタンが押されたとき，どのような処理を行っているのか順を追って説明していきましょう．

6．1　0番地スタート

電源投入時，リセットボタンを押したとき，RST 0を実行させたとき，これらの動作が行われたときはいずれも0番地から，モニタがスタートします．0番地からモニタがスタートすると，まず，モニタの使用しているRAMエリアつまりワーキングエリアのテストメモリを行います．ワーキングエリアのRAMが異常な場合は，HLT命令を実行し，CPUを止めてしまいますが，正常であることが確認できた場合は，次のようなイニシャライズを行います．

まず，8255の動作モードをPORT A，PORT Bを入力，PORT Cを出力に指定します．次に83C9H番地から83F7H番地までを0クリアし，83C9H番地から83CEH番地までのIN命令，OUT命令エリアにそれぞれのインストラクションコードおよびRETコードを格納します．また，CPUレジスタセーブエリアのうち，スタックポインタのセーブエリアに最初のユーザースタックとなる番地8391Hを格納します．

以上のイニシャライズが終了すると，0クリアされているディスプレイレジスタの内容を，LEDディスプレイに表示し，キー入力待ちになります．

6．2　TRAP割り込み処理

キーボードスイッチの一つに MON キーがあります．このキーを押すとモニタのWARM STARTを行うことができます．この MON キーの処理にはTRAP割り込みを使用しています．

TRAP割り込みは，他の割り込みがイネーブル状態であろうとなかろうと，それには無関係に割り込みがかかります．したがって MON キーを押せば，モニタ内部でもユーザープログラム内部でも割り込みがかかります．そこでモニタではTRAP割り込みの処理を次のようにしています．

トラップフラグの項でも述べましたが，CPU の処理がモニタ内部なのか，ユーザープログラム内部なのかを判断するのにトラップフラグを用いています．その結果ユーザープログラム実行中に［MON］キーが押されたと判断した場合は，［MON］キーが押されたときの CPU レジスタの内容を CPU レジスタセーブエリアに退避します．次に8255の動作モードを再設定し，83C9H 番地から 83F7H 番地までの CPU レジスタセーブエリア以外のワーキングエリアをイニシャライズします．また，退避されたプログラムカウンタおよび AF の表示処理を行い，キー入力ルーチンに入ります．つまりユーザープログラム実行中に［MON］キーを押せばモニタの WARM START を行うことができます．

モニタ内部で［MON］キーが押されると，CPU レジスタの退避は行わず，8255の動作モードの再設定および 83C9H 番地から 83F7H 番地までの CPU レジスタセーブエリア以外のワーキングエリアをイニシャライズし，0クリアされているディスプレイレジスタの内容を LED ディスプレイに表示し，キー入力待ちになります．

ユーザープログラム暴走中 WARM START を行うと，スタックポインタが異常な値になることが往々にしてあります．WARM START を行ったときはスタックポインタを確認し，もしそれが異常な値になっている場合は，常にスタックポインタを修正し直すように心がけて下さい．

モニタはモニタスタックエリアのほかに，一部ユーザースタックエリアを使用しています．したがってスタックポインタが異常な値のままですとモニタが正常に動かない場合があります．

6．3　RST 7.5 割り込み処理

ステップスイッチを STEP 側に倒すと，割り込みイネーブル状態（ユーザープログラム実行中）のとき，1命令実行する毎に RST 7.5 割り込みがかかります．割り込みがかかった後の処理は，ブレイクポイント，ブレイクデプスの内容により異なっています．

RST 7.5 の割り込み処理ルーチンに入ると，まず CPU レジスタの退避を行います．次にブレイクデプスの内容を調べ，それが0の場合はプログラムカウンタおよび AF を表示し，モニタのキー入力ルーチンに入ります．ブレイクデプスが0でない場合は，ブレイクポイントとプログラムカウンタの比較を行います．両者が一致しない場合はブレイクデプスをデクリメントし，再び0との比較を行います．0のときは先程と同様にプログラムカウンタおよび AF を表示し，モニタのキー入力ルーチンに入ります．0でないときは退避したレジスタの内容を復帰し再びユーザープログラムを実行します．

ブレイク動作を行いながらユーザープログラムを実行し，その途中で［MON］キーを押すと，PC，AF を表示せずに0を表示します．ブレイク動作を行うときはユーザープログラムを1命令実行する毎に RST 7.5 の割り込みがかかり，そのたびにモニタ処理を行っています．したがって一見ユーザープログラムのみを実行しているようですが，実際はモニタ処理がその大半を占めているのです．TRAP 処理で述べましたが，CPU がモニタを実行しているか，ユーザープログラムを実行しているかはトラップフラグによって判断しています．RST 7.5 の割り込みがかかると，ト

ラップフラグをモニタ側にセットしますが，トラップフラグをモニタ側にセットするまでにいくつかのインストラクションを実行しています．したがってブレイク動作を行っているときにTRAP割り込みをかけた場合，モニタ内部のアドレスを表示する可能性もあります．

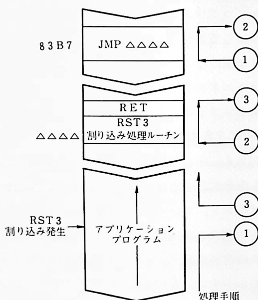
6．4　RST 6.5 割り込み処理

TK-85 では直接この割り込みを使用していません．RST 6.5 はTK-80BS のブレイク処理に用いられています．したがってモニタは，RST 6.5 の割り込みがかかったとき，TK-80BS のブレイクエントリーエリア，F125H番地にジャンプするようにしています．

6．5　リスタートジャンプテーブル

TK-85 モニタにおいて7種類のリスタート命令および1本の割り込み入力を開放しています．これらのリスタート命令を実行すると，おのおの次に示す番地に無条件にジャンプしていきます．したがってこのエリアに各処理ルーチンへのジャンプ命令を書き込んでおくことにより各処理を実行することができます．

RST 1	83B1H番地
RST 2	83B4H番地
RST 3	83B7H番地
RST 4	83BAH番地
RST 5	83BDH番地
RST 6	83C3H番地
RST 7	83C6H番地
RST 5.5	83C0H番地



7．　モードファンクションの説明

7．1　IN モード，OUT モード

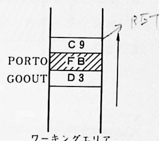
これら二つのモードファンクションにより，モニタ操作で直接ポートへデータを出力したり，ポートへの入力データを監視したりすることができます．

ワーキングエリアに IN 命令エリア，OUT 命令エリアがあり，モニタは直接ここに各インストラクションコード等を入れて，それぞれの命令を実行しています．しかしなぜこのようなことを行うのか疑問に思われる方がいるかもしれません．しかし，これにはそれなりの理由があるのです．

μCOM-85 あるいは μCOM-80 のインストラクション IN および OUT を参照して下さい．両者共オペランドはイミーディエイトデータになっています．つまりポートアドレスを可変にすることができないのです．したがって IN および OUT 命令を実行するときには，INおよび OUT ポートアドレスをワーキングエリアの RAM 上に格納することによって，ポートアドレスの可変化を図っているのです．

ポートアドレスが固定の場合　　ポートアドレスが可変の場合

```
        ⋮                        ⋮
   LDA     DATE            MVI     A, 0FBH
   OUT     0FBH            STA     PORT0
        ⋮                        ⋮
                           LDA     DATE
                           CALL    GOOUT
                                 ⋮
```



7．2　MOVE モード

MOVE モードはデータのブロック転送を行います．その用途は実にバラエティに富んでいますが，ここでは特にそのことには触れず，MOVE モードのアルゴリズムについて説明します．

データのブロック転送を行う場合，注意すべき事項があります．それはソースデータのエリアとディスティネーションデータのエリアが重なった場合，未転送のソースデータを破壊してしまう恐れがあるということです．図８．２の例を見て下さい．これは8000Ｈ番地から8100Ｈ番地までのデータ（ソースデータ）を80C0H 番地以降に8000Ｈ番地から順に転送しようとしている図です．この場合，80Ｃ0Ｈ番地から8100番地までのソースデータを破壊してしまい8180Ｈ番地から81C0H 番地までのディスティネーションデータのエリアには，その破壊されたデータが格納されてしまいます．

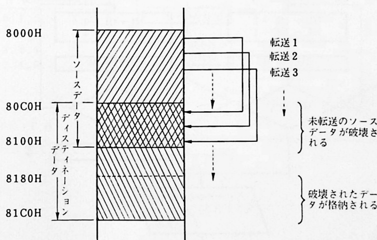


図8.2

HL≧DEの場合

ディスティネーションデータ
ソースデータ
1
2
3
DE
HL
BC

HL＜DEの場合

ソースデータ
ディスティネーションデータ
DE
BC ──→HL
3
2
1

修正前　修正後
HL ──→BC
DE



図8.3

入

BC←ソースデータエンドアドレス
DE←ディスティネーションスタートアドレス
HL←ソースデータスタートアドレス

HL：DE

HL≧DE

HL－HL－1
DE←DE－1

HL←HL＋1
HL←DE＋1

(DE)←M

HL＝BC

No

Yes

HL＜DE

PUSH B

HL↔BC

DE←DE＋HL－BC

POP H
PUSH D

HL←HL＋1
DE←DE＋1

HL←HL－1
DE←DE－1

(DE)←M

HL＝BC

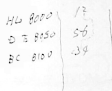
No

Yes

POP D

ディスティネーション
エンドアドレスDE
表示

出



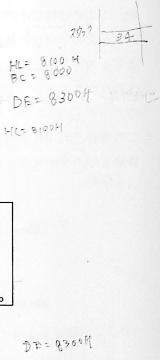


図8.4

以上のような現象を避けるために，ソースデータとディスティネーションデータの先頭アドレスの比較を行い，その大小によって転送順序を変えています．

ソースデータの先頭アドレスが，ディスティネーションデータの先頭アドレスに等しいか大きいときは，ソースデータの先頭アドレスから END アドレスの方に向ってデータを転送していきます．またソースデータの先頭アドレスがディスティネーションデータの先頭アドレスより小さいときは，ソースデータの END アドレスから先頭アドレスの方に向って転送していきます．

では，モニタがどのように処理しているか，転送部分のアルゴリズムを図8．3に示します．なお，HL レジスタにはソースデータの先頭アドレス，DE レジスタにはディスティネーションデータの先頭アドレス，BC レジスタにはソースデータの END アドレスが入っているものとします．

7．3　TEST MEMORY モード

テストメモリは，増設 RAM あるいは実装 RAM ユーザーズエリアの READ WRITE 動作の確認およびアドレスバス，データバスの誤配線等のチェックを行うときに使用します．

実際のテストは，次のような書き込み動作と読み込み動作を繰り返しています．

まず指定されたテスト領域に先頭から順に０，１，２，……というデータを書き込みます．次に書き込みデータ，つまり期待値と，RAM からの読み込みデータとの比較を行い正常な READ WRITE 動作が行われたかどうかの確認を行います．正常な場合は，再びテスト領域に先頭から順に１，２，３，……というデータを書き込み，上記のようなチェック動作を繰り返します．この書き込み，読み込み動作はテストデータが０Hから FFH まで計256回繰り返し行われます．

	書き込みデータ (HEXA)
領域 / 書き込み	STAT ——→ END
1	0，1，2，‥FF，0‥
2	1，2，3，…………
3	2，3，4，
4	3，4，5，
.	.
.	.
.	.
255	FE，FF，0………
256	FF，0，1………

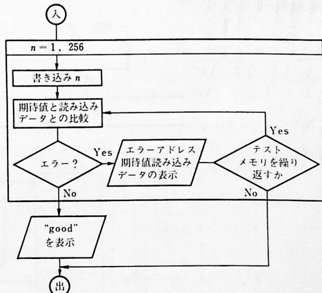


図8.5

READ，WRITE の動作の確認およびデータバスの誤配線を調べるときは，RAM 領域1バイト毎に読み書きを行うことで動作確認は十分なのですが，アドレスバス誤配線があった場合，それだけではチェックすることができません．なぜなら異常アドレスに，テストデータを書き込んでも，その異常アドレスからデータを読み込んでしまうからです．したがって，全テスト領域にテストデータを書き込んでからチェックを始めるという方法をとっています．では実際にどうなるのか，図8．6のような3本のアドレス線からなる RAM を想定して説明しましょう．

3本のアドレス線ですから，選択できるアドレスは0番地から7番地までです．ところが図8．6のような誤配線があった場合，2番地から5番地までは，他のアドレスが選択されてしまいます．したがって0番地から7番地まで書き込み1を行うと実際には，0番地に4，1番地に5，6番地に6，7番地に7というデータが書き込まれ，その他のアドレスには，書き込みが行われません．またこの後，0番地から7番地までの読み込み動作を行うと，4，5，6，7，4，5，6，7というデータが読み込まれてしまいます．書き込み1による期待値は0，1，2，3，4，5，6，7ですから，これでアドレスの誤配線が確認できるわけです．

テストメモリを行うと，その領域の RAM の内容を破壊していまいます．したがってモニタワーキングエリアのテストメモリは行わないようにくれぐれも注意して下さい．なおワーキングエリアのテストメモリはモニタが走った時点ですでに行われています．

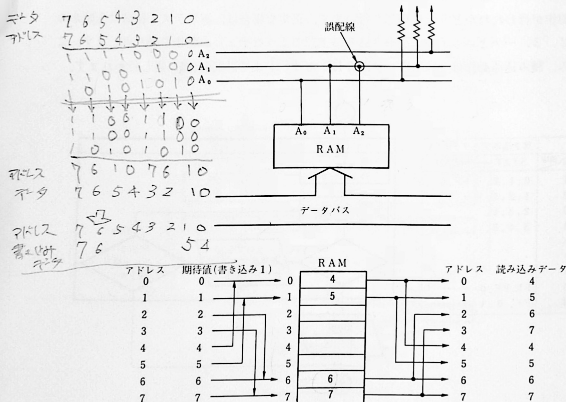


図8.6

7.4 SAVE モード，LOAD モード

μPD8085A には SIO および SOD というシリアルデータ入出力用の端子があります．

カセットテープへのファイル作成およびカセットテープからのファイルロードはすべてこの端子を利用しています．

SIO 端子からの入力データは，RIM 命令によってアキュムレータのビット 7 にロードされます．

またアキュムレータのビット 6 (SOD ENABLE) がセットされているとき，SIM 命令によってアキュムレータのビット 7 がSOD 出力ラッチにロードされます．

データ入出力はこのようにシリアルデータを扱います．データは周波数 2400Hz の波 2 周期をデータ“1”とし，周波数 1200Hz の波 1 周期をデータ“0”としています．したがってデータの転送速度は 1200 baud (1200bit/s) になっています．

データは 1 バイトを一つの単位としています．1 バイトの転送データは，スタートビット（“0”× 1）ではじまり，ストップビット（“1”× 2）で終了します．したがって 1 バイト転送に要するビット数は11，時間にして約9.17ms です．

○ 1 k バイトデータ構成 (23H)：SOD 出力波形

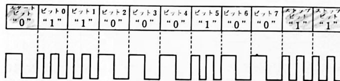


図8.7

1 バイトデータが前記のような形式になっていることが分かりました．カセットテープに生成させるファイルはこれら 1 バイトデータの集合である 1 プログラムが 1 単位となって構成されます．

各ファイルはファイル番号を有し，これによって各ファイルが判別されます．したがっていくつかのファイルがカセットテープにある場合も，これらファイル番号を基にファイル検索が可能になるわけです．

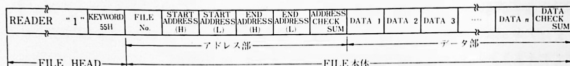
1 ファイルは，ファイルヘッド，アドレス部およびデータ部より構成されています．ファイルヘッドは READER 部および KEY WORD に分かれ，READER 部はデータ“1”6000ビットより成り，KEY WORD は 1 バイトデータ55H より成っています．

アドレス部は，1 バイトのファイル No., START ADDRESS HIGH, LOW および END ADDRESS HIGH, LOW そして，アドレスチェックサムより構成されています．1 バイトデータ構成によって気づかれたかと思いますが，1 バイトデータ中にパリティービットを設けておらずエラーがあっても認識できません．したがってアドレス部にエラーがあった場合，暴走によってワーキングエリア等を破壊してしまう恐れがあります．そこでアドレスエラー発生時点で LOAD モードを中断するためこのアドレスチェックサムを設けてあります．

データ部はその名の通り，１バイトデータの集まりです．このデータ部にもチェックサムを設けてあり，データロードが正常に行われたかどうかチェックしています．

なお詳しい波形生成方法および検出方法は10章のモニターサブルーチンを参照して下さい．

○ファイル構成



○ファイルヘッド



○ADDRESS CHEK SUM：アドレス部（ADDRESS CHECK SUMを除く）の和の２の補数
○DATA CHECK SUM　：データ部（DATA CHECK SUMを除く）の和の２の補数

図8.8

8．μPD8255 のプログラミング

μPD8255 は μPD8080A（μPD8085A）マイクロコンピュータシステム用に開発されたプログラマブル汎用Ｉ/Ｏデバイスです．３組（８ビット）の入出力ポートがあり，プログラムによる制御でデータ入力，データ出力，ステータス信号入力，コントロール信号出力に使用することができます．

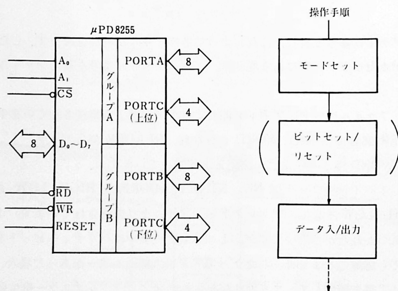


図8.9

μPD8255 はその使用によって次の三つのモードに分けられます．

モード 0：基本的な入出力ポート

モード 1：コントロール信号，ステータス信号による制御にともなう入出力ポート

モード 2：双方向データを扱う入出力ポート

モードの選択は CPU から μPD8255 に対して送られる 8 ビットのコントロールワードによって行います．

ここでは最も基本的に使われるモード 0 での使い方を説明します（図 8．9 参照）．

8255は RESET 信号を受けると，すべてのポートが入力モードに戻されて，以後命令で新しいモードがセットされない限り，この入力モードは継続されます．

モード 0 では基本的な入出力ポートとして動作します．CPU からコントロールワードを出力することによってポート A，ポート B，ポート C（上位），ポート C（下位）がそれぞれ独立に，入力ポートあるいは出力ポートに設定されます．

コントロールワードを μPD8255 に出力するときには，μPD8255 の各端子を次の状態にして出力しなければなりません．

$\overline{CS}$	A_0	A_1	$\overline{RD}$	$\overline{WR}$	動作	
0	1	1	1	0	データバス→制御	出力

ここで $\overline{RD}$ および $\overline{WR}$ はそのまま CPU の $\overline{RD}$ および $\overline{WR}$ に接続されているので入出力命令を使用するだけでそれぞれの機能を果し，特別ソフトウェアを組む上で考慮する必要はありません．

ここで問題になるのは $\overline{CS}$，A_1，A_0です．　8255のA_1およびA_0はアドレスバスのA_1およびA_0にそれぞれ接続されています．$\overline{CS}$ は図 8．10のようになっています．

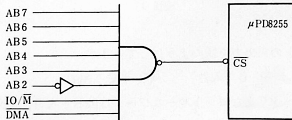


図8.10

これを見れば分かりますが，CS がアクティブになるのは次の条件が満たされたときです．まず DMA 転送時でないこと，リード/ライトが I/O に対して行われること，AB7～AB3 が High レベル（“1”）AB2 が LOW レベル（“0”）であることです．

この中でソフトウェアが関与するのは，アドレスバスの部分だけです．したがって $\overline{CS}$ をアクティブにするためには，アウトプットアドレスの内 AB7～AB3 を “1” AB2を “0” に設定す

ればよいわけです．

コントロールワードをμPD8255に出力するときには上記の $\overline{CS}$ をアクティブにする条件と A_1 および A_0 を考慮すればいいので，アウトプットアドレスは FBH(AB7～AB3＝“1”，AB2＝“0” AB1～AB0＝“1”）となり，命令は OUT 0FBH となります．

またこのアウトプットアドレスへの出力データつまりコントロールワードの仕様は図8.11のようになっています．

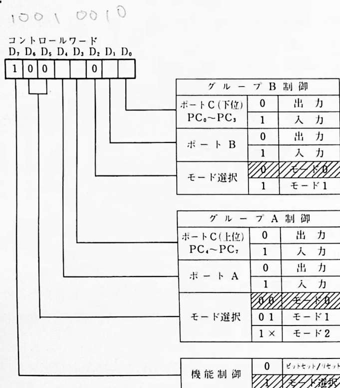


図8.11

TK-85では，各ポートの入出力を次のようにしています．

ポートA：入力　　ポートB：入力　　ポートC：出力

したがってμPD8255へ送り込むコントロールワードは92Hとなり，TK-85のモニタの最初で次のような命令を実行しています．

```
MVI  A, 92 H；アキュムレータにコントロールワードをロードする．
OUT  0FBH   ；μPD8255にコントロールワードを出力する．
```

コントロールワード送出後，各ポートとデータの入出力を行いますが，データを送り込む（アキュムレータの内容を各ポートへ送出）には OUT 命令を用い，各ポートからデータを入力する（各ポートからアキュムレータへロード）には IN 命令を用います．

このときのI/Oアドレスはポートによって異なり，具体的には図8.12のようになります．

コーディング		$\overline{CS}$										動作	
		A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	$\overline{RD}$	$\overline{WR}$		
IN	0F8H	1	1	1	1	1	0	0	0	0	1	ポート A→アキュムレータ	入力
IN	0F9H	1	1	1	1	1	0	0	1	0	1	ポート B→アキュムレータ	
IN	0FAH	1	1	1	1	1	0	1	0	0	1	ポート C→アキュムレータ	
OUT	0F8H	1	1	1	1	1	0	0	0	1	0	アキュムレータ→ポート A	出力
OUT	0F9H	1	1	1	1	1	0	0	1	1	0	アキュムレータ→ポート B	
OUT	0FAH	1	1	1	1	1	0	1	0	1	0	アキュムレータ→ポート C	

I/Q アドレス（A_7～A_0）

図8.12

μPD8255 のポートCが出力としてプログラムされているときは，そのビットの位置を指定して，1命令でセットまたはリセットすることができます．これは図 8.13のようなコントロールワードを，μPD8255 に送り込む（OUT 0FBH）ことによって実現されます．

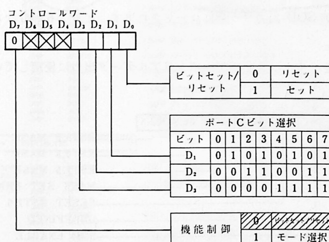


図8.13

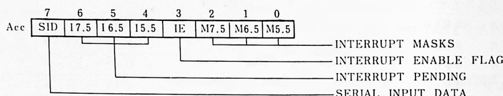
9．RIM, SIM について

μPD8085A には，μPD8080A に追加された命令が二つあります．それがこの RIM 命令と SIM 命令です．

RIM は read interrupt mask の略です．この命令を実行するとアキュムレータにリスタート割り込みマスク，保留割り込み，そして SID の内容がロードされます．

モニタはカセットテープからのシリアル入力に RIM 命令を用い SID 端子をシリアルデータの

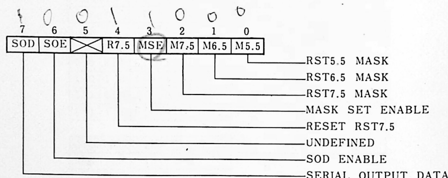
入力に使用しています．



SIM は set interrupt masks の略です．アキュムレータの内容がリスタート割り込みマスクのプログラミングのために用いられます．ビット 0 ～ 2 は，ビット 3 が 1 （セット）ならインターラプトマスクレジスタの RST 5.5, 6.5, 7.5 に対するマスクビットをセットまたはリセットします．ビット 3 は“マスクセット イネーブル”制御フラグです．マスクをセットすると（マスクビット＝1），関連のインターラプトは禁止されます．RST 7.5 はマスクされてもされなくてもビット 4 ＝ 1 ならセットされます．モニタは，ユーザープログラムにジャンプする時点ですべてのインターラプトマスクをリセットしています．

SIM は，SOD 出力ラッチのロードも行います．もしビット 6 がセットされていれば，アキュムレータのビット 7 が SOD 出力ラッチにロードされます．ビット 6 が 0 ならラッチは影響されません．

モニタはこの機能を，カセットテープへのシリアルデータ出力に使用しています．



9章　モニタプログラムリスト

```
E STNO ADRS OBJECT M   SOURCE STATEMENTS

                          TITLE   '     NEW TK-85  MONITOR'
                    ;********************************
                    ;**        NEW TK-85           **
                    ;**         MONITOR            **
                    ;**     '80.2.15   NEC         **
                    ;********************************
                    ;
                    ;
                    ;********************************
                    ;**   MONITOR START            **
                    ;********************************
                    ;
     0000                 ORG     0
     0000 3E92            MVI     A,CTRLW
     0002 D3FB            OUT     MODES          ;TRANSFER CONTROL WORD (<MODE 0> PA:IN PB:IN PC:OUT) TO 8255
     0004 C35F00          JMP     RESET          ;JUMP INITIALIZE AREA
     0007                 ORG     8
     0008 C3B183          JMP     RST1           ;JUMP RST1 PROCESSING AREA
     000B                 ORG     10H
     0010 C3B483          JMP     RST2           ;JUMP RST2 PROCESSING AREA
     0013                 ORG     18H
     0018 C3B783          JMP     RST3           ;JUMP RST3 PROCESSING AREA
     001B                 ORG     20H
     0020 C3BA83          JMP     RST4           ;JUMP RST4 PROCESSING AREA
     0023                 ORG     24H
     0024 C3D301          JMP     TRAP           ;JUMP MONITOR
     0027                 ORG     28H
     0028 C3BD83          JMP     RST5           ;JUMP RST5 PROCESSING AREA
     002B                 ORG     2CH
     002C C3C083          JMP     RST55          ;JUMP RST5.5 PROCESSING AREA
     002F                 ORG     30H
     0030 C3C383          JMP     RST6           ;JUMP RST6 PROCESSING AREA
     0033                 ORG     34H
     0034 C325F1          JMP     BSBRK          ;JUMP BREAK ENTRY AREA (TK-80 BS)
     0037                 ORG     38H
     0038 C3C683          JMP     RST7           ;JUMP RST7 PROCESSING AREA
     003B                 ORG     3CH
     003C C31902          JMP     STEP           ;JUMP BREAK ENTRY AREA (NEW TK-85)
                    ;
                    ;********************************
                    ;**   INITIALIZE               **
                    ;********************************
                    ;
     003F 21C983    CLEAR: LXI    H,GOIN         ;SET TOP ADD. OF CLEAR AREA
     0042 AF        CLEA1: XRA    A
     0043 77        CLEA2: MOV    M,A            ;CLEAR WORKING AREA
     0044 23              INX     H
     0045 05              DCR     B
     0046 C24300          JNZ     CLEA2
     0049 32DC83          STA     FTRAP          ;SET TRAP FLAG TO SYSTEM (0)
     004C 3EC9            MVI     A,0C9H
     004E 32CE83          STA     PORTO+1        ;INITIALIZE OUT INSTRUCTION AREA
     0051 32CB83          STA     PORTI+1        ;INITIALIZE  IN INSTRUCTION AREA
     0054 3ED3            MVI     A,0D3H
     0056 32CC83          STA     GOOUT          ;SET OUT INSTRUCTION CODE
     0059 3EDB            MVI     A,0DBH
     005B 32C983          STA     GOIN           ;SET  IN INSTRUCTION CODE
     005E C9              RET
                    ;
     005F 219183    RESET: LXI    H,USESP        ;SET TOP ADD. OF MONITORS WORKING AREA
     0062 7E        RESE1: MOV    A,M
     0063 2F              CMA
     0064 77              MOV     M,A
     0065 BE              CMP     M
     0066 CA6A00          JZ      RESE2
     0069 76              HLT                    ;IF WORKING AREA ERROR THEN HALT
     006A 2C        RESE2: INR    L
     006B C26200          JNZ     RESE1
                    ;
     006E 31B183          LXI     SP,RST1        ;SET TOP ADD. OF MONITORS STACK
     0071 062F            MVI     B,DIG-GOIN     ;SET CLEAR COUNTER
     0073 CD3F00          CALL    CLEAR
     0076 219183          LXI     H,USESP
     0079 22E283          SHLD    SVSP           ;MOVE TOP ADD. OF USERS STACK TO (SP)SAVING AREA
                    ;
                    ;********************************
                    ;**   DISTRIBUTE FUNCTION      **
                    ;********************************
                    ;
     007C CD6C05    START: CALL   SEGCV          ;SEGMENT DATA CONVERSION
     007F CD8406          CALL    KEYIN          ;KEY INPUT (A,B:HEXA DATA)
                    ;
     0082 E610      STAT1: ANI    10H
     0084 CABB00          JZ      DIGIT          ;IF INPUT DATA IS EQUAL TO DIGIT THEN JUMP DIGIT PROCESS
```

```
                  ;
0087 3ADF83                LDA     FMODE
008A OF                    RRC
008B DA9500                JC      STAT2           ;IF F-MODE IS ON THEN SELECT INPUT DATA
                  ;
008E 3ADE83                LDA     FREG
0091 OF                    RRC
0092 D29B00                JNC     STAT3           ;IF F-REG IS ON THEN SELECT INPUT DATA
                  ;
0095 78           STAT2:   MOV     A,B
0096 FE16                  CPI     16H
0098 DA7C00                JC      START           ;IF INPUT DATA IS EQUAL TO REG OR MODE THEN EXECUTE THIS ONE
                  ;
009B 78           STAT3:   MOV     A,B
009C E60F                  ANI     OFH
009E 0600                  MVI     B,O
00A0 87                    ADD     A
00A1 4F                    MOV     C,A
00A2 21AB00                LXI     H,TFUNC
00A5 09                    DAD     B
00A6 7E                    MOV     A,M
00A7 23                    INX     H
00A8 66                    MOV     H,M             ;SET H.ADD. OF JUMPING ROUTINE
00A9 6F                    MOV     L,A             ;SET L.ADD. OF JUMPING ROUTINE
00AA E9                    PCHL                    ;JUMP EACH FUNCTION KEY PROCESS
                  ;
00AB 9701         TFUNC:   DW      RUN
00AD A401                  DW      CONT
00AF 6601                  DW      ADRST
00B1 8D01                  DW      RDDEC
00B3 8301                  DW      RDINC
00B5 7902                  DW      WTENT
00B7 DA00                  DW      MDFKP
00B9 E900                  DW      REG
                  ;
                  ;*******************************
                  ;**    DIGIT PROCESS         **
                  ;*******************************
                  ;
00BB 3ADE83       DIGIT:   LDA     FREG
00BE OF                    RRC
00BF DA1201                JC      REGFN           ;IF F-REG IS ON THEN DISPLAY REGISTER
                  ;
00C2 3ADF83                LDA     FMODE
00C5 OF                    RRC
00C6 DAF200                JC      MD              ;IF F-MODE IS ON THEN SET EACH MODE FUNCTION FLAG
                  ;
00C9 CDD205                CALL    SHIFT           ;SHIFT (DR) & (DDW)
00CC 3AEC83                LDA     DATA
00CF B0                    ORA     B
00D0 32EC83                STA     DATA            ;INPUT DIGIT DATA (DR)
00D3 78                    MOV     A,B
00D4 32CF83                STA     DDW4            ;INPUT DIGIT DATA (DDW)
00D7 C37C00                JMP     START
                  ;
                  ;*******************************
                  ;**  FLAG(MODE&REG) PROCESS  **
                  ;*******************************
                  ;
00DA CDE905       MDFKP:   CALL    FLAGC           ;RESET EACH FLAG
00DD 32DF83                STA     FMODE           ;SET F-MODE
00E0 CD2906       ALCLA:   CALL    CA              ;CLEAR (ADW) & (AR)
00E3 CD3906       DTCLA:   CALL    CD              ;CLEAR (DDW) & (DR)
00E6 C37C00                JMP     START
                  ;
00E9 CDE905       REG:     CALL    FLAGC           ;RESET EACH FLAG
00EC 32DE83                STA     FREG            ;SET F-REG
00EF C3E000                JMP     ALCLA
                  ;
                  ;*******************************
                  ;** MODE FUNCTION KEY PROCESS**
                  ;*******************************
                  ;
00F2 78           MD:      MOV     A,B
00F3 FE0A                  CPI     OAH
00F5 DA7C00                JC      START           ;IF INPUT DATA IS NOT EQUAL TO MODE FUNCTION THEN INPUT AGAIN
                  ;
00F8 D608                  SUI     8
00FA CDF505                CALL    STBIT           ;SET BIT
00FD 32DF83                STA     FMODE           ;SET EACH MODE FLAG BIT (RESET F-MODE)
0100 E6C0                  ANI     OCOH
0102 CA7C00                JZ      START           ;IF F-IN OR OUT IS ON THEN IN/OUT PROCESS
                  ;
0105 17                    RAL
0106 17                    RAL
0107 32D683                STA     ADW1            ;GET 'O' OR 'I'
010A 3E1D                  MVI     A,1DH
010C 32D583                STA     ADW2            ;GET '-'
010F C37C00                JMP     START
                  ;
                  ;*******************************
                  ;**  DISPLAY REGISTER        **
                  ;*******************************
                  ;
0112 78           REGFN:   MOV     A,B
0113 FE07                  CPI     7
0115 D27C00                JNC     START           ;IF INPUT DATA IS GREATER THAN 6 THEN START
                  ;
0118 CDF505                CALL    STBIT           ;SET BIT
011B 07                    RLC
011C 32DE83                STA     FREG            ;SET EACH REGISTER FLAG BIT (RESET F-REG)
                  ;
011F 78                    MOV     A,B
```

```
0120 0600                  MVI     B,0
0122 87                    ADD     A
0123 4F                    MOV     C,A
0124 214A01                LXI     H,TRGNM
0127 09                    DAD     B
0128 7E                    MOV     A,M
0129 23                    INX     H
012A 66                    MOV     H,M
012B 6F                    MOV     L,A
012C 22D583                SHLD    ADW2            ;DISPLAY REGISTER NAME
                   ;
012F 215801                LXI     H,TRGAD
0132 09                    DAD     B
0133 7E                    MOV     A,M
0134 23                    INX     H
0135 66                    MOV     H,M             ;SET H.ADD. OF REGISTER SAVING AREA
0136 6F                    MOV     L,A             ;SET L.ADD. OF REGISTER SAVING AREA
0137 7E                    MOV     A,M
0138 23                    INX     H
0139 66                    MOV     H,M             ;SET (H.ADD. OF REGISTER SAVING AREA)
013A 6F                    MOV     L,A             ;SET (L.ADD. OF REGISTER SAVING AREA)
013B 22EC83                SHLD    DATA            ;SET (REGISTER SAVING AREA)
                   ;
013E 211D1D                LXI     H,1D1DH
0141 22D383                SHLD    ADW4            ;GET "--"
0144 CD6406                CALL    DWCH            ;CONVERT (DR) INTO (DDW)
0147 C37C00                JMP     START
                   ;
014A 0F0A          TRGNM:  DW      0A0FH
014C 0C0B                  DW      0B0CH
014E 0E0D                  DW      0D0EH
0150 1210                  DW      1012H
0152 1505                  DW      0515H
0154 150B                  DW      0B15H
0156 0D0B                  DW      0B0DH
                   ;
0158 EA83          TRGAD:  DW      SVAF
015A E883                  DW      SVBC
015C E683                  DW      SVDE
015E E483                  DW      SVHL
0160 E283                  DW      SVSP
0162 F083                  DW      SVBP
0164 F283                  DW      SVBD
                   ;
                   ;********************************
                   ;**   ADDRESS SET PROCESS      **
                   ;********************************
                   ;
0166 CD7C01        ADRST:  CALL    ARSET           ;MOVE (DR) TO (AR)
                   ;
0169 7E            MEMRD:  MOV     A,M
016A 32EC83                STA     DATA            ;READ MEMORY
016D CD5106                CALL    AWCH            ;CONVERT (AR) INTO (ADW)
0170 CD4306                CALL    CD12            ;CLEAR (DR)H & (DDW1,2)
0173 CD6D06                CALL    DLWCH           ;CONVERT (DR)L INTO (DDW3,4)
0176 CDE905                CALL    FLAGC           ;RESET EACH FLAG
0179 C37C00                JMP     START
                   ;
017C 2AEC83        ARSET:  LHLD    DATA
017F 22EE83                SHLD    ADRES
0182 C9                    RET
                   ;
                   ;********************************
                   ;**   READ INCREMENT PROCESS **
                   ;********************************
                   ;
0183 2AEE83        RDINC:  LHLD    ADRES
0186 23                    INX     H               ;INCREMENT (AR)
0187 22EE83                SHLD    ADRES
018A C36901                JMP     MEMRD
                   ;
                   ;********************************
                   ;**   READ DECREMENT PROCESS **
                   ;********************************
                   ;
018D 2AEE83        RDDEC:  LHLD    ADRES
0190 2B                    DCX     H               ;DECREMENT (AR)
0191 22EE83                SHLD    ADRES
0194 C36901                JMP     MEMRD
                   ;
                   ;********************************
                   ;**   RUN PROCESS              **
                   ;********************************
                   ;
0197 2AEE83        RUN:    LHLD    ADRES
019A 22E083                SHLD    SVPC            ;MOVE (AR) TO (PC)SAVING AREA
019D C3A401                JMP     CONT
                   ;
                   ;  <<<  TK-80 MONITOR SUBROUTINE "RGDSP"  >>>
                   ;
01A0                       ORG     01A1H
01A1 C3CF07                JMP     RGDSP
                   ;
                   ;********************************
                   ;**   CONTINUE PROCESS         **
                   ;********************************
                   ;
01A4 2AE283        CONT:   LHLD    SVSP
01A7 F9                    SPHL                    ;SET ((SP)SAVING AREA)
01A8 2AE083                LHLD    SVPC
01AB E5                    PUSH    H               ;SAVE ((PC)SAVING AREA)
01AC 2AEA83                LHLD    SVAF
01AF E5                    PUSH    H               ;SAVE ((AF)SAVING AREA)
```

```
01B0 2AE483          LHLD   SVHL
01B3 E5              PUSH   H             ;SAVE ((HL)SAVING AREA)
            ;
01B4 2AE883          LHLD   SVBC
01B7 E5              PUSH   H             ;SAVE ((BC)SAVING AREA)
01B8 C1              POP    B             ;RECOVER BC
01B9 2AE683          LHLD   SVDE
01BC EB              XCHG                 ;RECOVER DE
01BD C3C301          JMP    CONT1
            ;
            ;  <<<  TK-80 MONITOR SUBROUTINE "SEGCG"  >>>
            ;
01C0                 ORG    01C0H
01C0 C3DF07          JMP    SEGCG
            ;
01C3 E1     CONT1:   POP    H             ;RECOVER HL
            ;
01C4 AF              XRA    A
01C5 D3FA            OUT    PORTC         ;RESET EXTERNAL F/F WHICH IS IN CONTACT WITH RST7.5 F/F (PC7
01C7 3D              DCR    A
01C8 32DC83          STA    FTRAP         ;SET TRAP FLAG TO USER (FFH)
01CB 3E98            MVI    A,MSKRS
01CD 30              SIM                  ;RESET INTERRUPT MASK
01CE D3FA            OUT    PORTC         ;INACTIVATE 'PRESET' OF EXTERNAL F/F (PC7)
            ;
01D0 F1              POP    PSW           ;RECOVER AF
01D1 FB              EI                   ;INTERRUPT ENABLE
01D2 C9              RET                  ;RECOVER PC
            ;
            ;********************************
            ;**  TRAP INTERRUPT PROCESS  **
            ;********************************
            ;
01D3 F5     TRAP:    PUSH   PSW           ;SAVE AF
01D4 3ADC83          LDA    FTRAP
01D7 A7              ANA    A
01D8 C2E401          JNZ    TRAP1         ;IF TRAP FLAG IS SYSTEM THEN REGISTERS ARE NOT SAVED
            ;
01DB 31B183          LXI    SP,RST1       ;SET TOP ADD. OF MONITORS STACK
01DE CD0202          CALL   INIT
01E1 C37C00          JMP    START
            ;
01E4 22E483 TRAP1:   SHLD   SVHL          ;MOVE HL TO (HL)SAVING AREA
01E7 210400          LXI    H,4
01EA 39              DAD    SP
01EB 22E283          SHLD   SVSP          ;MOVE SP (BEFORE OCCURRENCE OF TRAP) TO (SP)SAVING AREA
01EE F1              POP    PSW           ;RECOVER AF
01EF E1              POP    H             ;RECOVER PC
01F0 22E083          SHLD   SVPC          ;MOVE SAVED PC (CURRENT PC OF USERS) TO (PC)SAVING AREA
01F3 31EC83          LXI    SP,DATA       ;SET TOP ADD. OF CPU REGISTER SAVING AREA
01F6 F5              PUSH   PSW           ;MOVE AF TO (AF)SAVING AREA
01F7 C5              PUSH   B             ;MOVE BC TO (BC)SAVING AREA
01F8 D5              PUSH   D             ;MOVE DE TO (DE)SAVING AREA
01F9 31B183          LXI    SP,RST1       ;SET TOP ADD. OF MONITORS STACK
01FC CD0202          CALL   INIT
01FF C35D02          JMP    BRET
            ;
0202 3E92   INIT:    MVI    A,CTRLW
0204 D3FB            OUT    MODES         ;TRANSFER CONTROLE WORD(<MODE 0> PA:IN PB:IN PC:OUT) TO 8255
0206 0617            MVI    B,SVPC-GOIN   ;SET CLEAR COUNTER
0208 CD3F00          CALL   CLEAR
020B 21EC83          LXI    H,DATA        ;SET TOP ADD. OF CLEAR AREA
020E 060C            MVI    B,DIG-DATA    ;SET CLEAR COUNTER
0210 C34200          JMP    CLEA1
            ;
            ;  <<<  TK-80 MONITOR SUBROUTINE "KEYIN"  >>>
            ;
0213                 ORG    0216H
0216 C38406          JMP    KEYIN
            ;
            ;********************************
            ;** RST7.5 INTERRUPT PROCESS **
            ;********************************
            ;
0219 E3     STEP:    XTHL                 ;EXCHANGE (HL) FOR SAVED PC
021A 22E083          SHLD   SVPC          ;MOVE SAVED PC (CURRENT PC OF USERS) TO (PC)SAVING AREA
021D F5              PUSH   PSW           ;SAVE F
021E C32602          JMP    STEP1
            ;
            ;  <<<  TK-80 MONITOR SUBROUTINE "INPUT"  >>>
            ;
0221                 ORG    0223H
0223 C39006          JMP    INPUT
            ;
0226 210400 STEP1:   LXI    H,4
0229 39              DAD    SP
022A 22E283          SHLD   SVSP          ;MOVE SP (BEFORE OCCURRENCE OF RST7.5) TO (SP)SAVING AREA
022D F1              POP    PSW           ;RECOVER AF
022E E1              POP    H             ;RECOVER HL
022F 31EC83          LXI    SP,DATA       ;SET TOP ADD. OF CPU REGISTER SAVING AREA
0232 F5              PUSH   PSW           ;MOVE AF TO (AF)SAVING AREA
0233 C5              PUSH   B             ;MOVE BC TO (BC)SAVING AREA
0234 D5              PUSH   D             ;MOVE DE TO (DE)SAVING AREA
0235 E5              PUSH   H             ;MOVE HL TO (HL)SAVING AREA
0236 31B183          LXI    SP,RST1       ;SET TOP ADD. OF MONITORS STACK
            ;
0239 AF              XRA    A
023A 32DC83          STA    FTRAP         ;SET TRAP FLAG TO SYSTEM (0)
023D CD6F02          CALL   BRDPT
0240 CA5D02          JZ     BRET          ;IF (BD) IS EQUAL TO 0 THEN DISPLAY PC,AF AND JUMP MONITOR
            ;
0243 2AF083          LHLD   SVBP
0246 EB              XCHG                 ;DE : BREAK POINT
```

```
0247 2AE083           LHLD    SVPC           ;HL : PC
024A CDFF05           CALL    COMPA          ;IF (PC) IS EQUAL TO (BP) THEN SET ZERO FLAG ELSE RESET ONE
024D C2A401           JNZ     CONT           ;IF (PC) IS NOT EQUAL TO (BP) THEN CONTINUE
0250 2AF283           LHLD    SVBD
0253 2B               DCX     H              ;IF (PC) IS EQUAL TO (BP) THEN DECREMENT (BD)
0254 22F283           SHLD    SVBD
0257 CD6F02           CALL    BRDPT
025A C2A401           JNZ     CONT           ;IF (BD) IS NOT EQUAL TO 0 THEN CONTINUE
                   ;
025D 2AE083   BRET:   LHLD    SVPC
0260 22EE83           SHLD    ADRES          ;MOVE (PC) TO (AR)
0263 2AEA83           LHLD    SVAF
0266 22EC83           SHLD    DATA           ;MOVE (AF) TO (DR)
0269 CD4E06           CALL    CH             ;CONVERT (AR),(DR) INTO (ADW),(DDW)
026C C37C00           JMP     START
                   ;
026F 3AF283   BRDPT:  LDA     SVBD
0272 A7               ANA     A
0273 C0               RNZ
0274 3AF383           LDA     SVBD+1
0277 A7               ANA     A
0278 C9               RET
                   ;
                   ;*******************************
                   ;**  WRITE ENTER KEY PROCESS **
                   ;*******************************
                   ;
0279 3ADF83   WTENT:  LDA     FMODE
027C A7               ANA     A
027D C2BF02           JNZ     PRMOD          ;IF MODE FLAG IS ON THEN JUMP EACH MODE FUNCTION PROCESS
                   ;
0280 3ADE83           LDA     FREG
0283 A7               ANA     A
0284 C29502           JNZ     PRREG          ;IF REG FLAG IS ON THEN MODIFY REGISTER AND DISPLAY NEXT ONE
                   ;
0287 2AEE83           LHLD    ADRES
028A 3AEC83           LDA     DATA
028D 77               MOV     M,A            ;WRITE MEMORY
028E BE               CMP     M              ;READ AFTER WRITE
028F C21706           JNZ     ERR            ;IF WE COULD NOT WRITE DATA TO MEMORY THEN DISPLAY E-MESSAGE
0292 C38301           JMP     RDINC
                   ;
                   ;*******************************
                   ;**   MODIFY REGISTER        **
                   ;*******************************
                   ;
0295 215801   PRREG:  LXI     H,TRGAD        ;SET TOP ADD. OF SAVING REGISTER ADD.'S TABLE
0298 0601             MVI     B,1            ;SET POINTER OF NEXT REGISTER
029A 3ADE83           LDA     FREG           ;GET REGISTER FLAG
029D 0F               RRC
029E 0F       RSIFT:  RRC
029F DAA802           JC      DTSET          ;IF EACH REGISTER FLAG BIT IS ON THEN MODIFY ITS DATA
02A2 23               INX     H
02A3 23               INX     H
02A4 04               INR     B
02A5 C39E02           JMP     RSIFT
                   ;
02A8 7E       DTSET:  MOV     A,M
02A9 23               INX     H
02AA 66               MOV     H,M
02AB 6F               MOV     L,A
02AC 3AEC83           LDA     DATA
02AF 77               MOV     M,A            ;MODIFY (REGISTER SAVING AREA)L
02B0 23               INX     H
02B1 3AED83           LDA     DATA+1
02B4 77               MOV     M,A            ;MODIFY (REGISTER SAVING AREA)H
02B5 78               MOV     A,B
02B6 EE07             XRI     7
02B8 C21201           JNZ     REGFN
02BB 47               MOV     B,A            ;IF B IS EQUAL TO 7 THEN INITIALIZE B BECAUSE OF REPEATING
02BC C31201           JMP     REGFN
                   ;
                   ;*******************************
                   ;** JUMP EACH MODE FUNCTION **
                   ;*******************************
                   ;
02BF 07       PRMOD:  RLC
02C0 DAD602           JC      FIN            ;JUMP  IN  PROCESS
02C3 07               RLC
02C4 DAFF02           JC      FOUT           ;JUMP  OUT PROCESS
02C7 07               RLC
02C8 DAA203           JC      FMOV           ;JUMP MOVE PROCESS
02CB 07               RLC
02CC DA2203           JC      FTM            ;JUMP  TM  PROCESS
02CF 07               RLC
02D0 DADC04           JC      FLOAD          ;JUMP LOAD PROCESS
02D3 C32504           JMP     FSAVE          ;JUMP SAVE PROCESS
                   ;
                   ;*******************************
                   ;**   IN PROCESS             **
                   ;*******************************
                   ;
02D6 3ADB83   FIN:    LDA     ENTCT
02D9 A7               ANA     A
02DA C2ED02           JNZ     PIN            ;IF ENTCNT IS NOT EQUAL TO 0 THEN EXECUTE IN INSTRUCTION
                   ;
02DD 3C               INR     A
02DE 32DB83           STA     ENTCT          ;INCREMENT ENTCNT
02E1 3AEC83           LDA     DATA
02E4 32CA83           STA     PORTI          ;SET IN PORT ADD.
02E7 CD7706           CALL    WORDC
02EA 22D383           SHLD    ADW4           ;SET IN PORT ADD.
                   ;
```

```
02ED CDC983   PIN:    CALL    GOIN         ;EXECUTE IN INSTRUCTION
02F0 32EC83           STA     DATA         ;SET INPUT DATA
02F3 CD7706           CALL    WORDC
02F6 22CF83           SHLD    DDW4         ;SET INPUT DATA
02F9 CD4306   OUTIN:  CALL    CD12         ;CLEAR (DDW1,2) & (DR)H
02FC C37C00           JMP     START
              ;
              ;********************************
              ;**  OUT PROCESS              **
              ;********************************
              ;
02FF 3ADB83   FOUT:   LDA     ENTCT
0302 A7               ANA     A
0303 C21903           JNZ     POUT         ;IF ENTCNT IS NOT EQUAL TO 0 THEN EXECUTE OUT INSTRUCTION
              ;
0306 3C               INR     A
0307 32DB83           STA     ENTCT        ;INCREMENT ENTCNT
030A 3AEC83           LDA     DATA
030D 32CD83           STA     PORTO        ;SET OUT PORT ADD.
0310 CD7706           CALL    WORDC
0313 22D383           SHLD    ADW4         ;SET OUT PORT ADD.
0316 C3E300           JMP     DTCLA
              ;
0319 3AEC83   POUT:   LDA     DATA         ;GET OUT DATA
031C CDCC83           CALL    GOOUT        ;EXECUTE OUT INSTRUCTION
031F C3F902           JMP     OUTIN
              ;
              ;********************************
              ;** TEST MEMORY PROCESS       **
              ;********************************
              ;
0322 3ADB83   FTM:    LDA     ENTCT
0325 A7               ANA     A
0326 C23903           JNZ     FTM1         ;IF ENTCNT IS NOT EQUAL TO 0 THEN FTM1
              ;
0329 3C               INR     A
032A 32DB83           STA     ENTCT        ;INCREMENT ENTCNT
032D CD7C01           CALL    ARSET        ;MOVE (DR) TO (AR)
0330 22D783           SHLD    SVDAT        ;SAVE START ADD.
0333 CD5106           CALL    AWCH         ;CONVERT (AR) INTO (ADW)
0336 C3E300           JMP     DTCLA
              ;
0339 AF       FTM1:   XRA     A
033A 4F               MOV     C,A          ;GET TEST DATA
033B 2AEC83           LHLD    DATA
033E EB               XCHG                 ;DE : END ADD.
              ;
033F 2AD783   FTM2:   LHLD    SVDAT        ;HL : START ADD.
0342 CD1206           CALL    ERRMS        ;IF START ADD. IS GREATER THAN END ADD. THEN ERROR
0345 41               MOV     B,C
0346 2B               DCX     H
0347 23       FTM3:   INX     H
0348 70               MOV     M,B          ;WRITE TEST DATA
0349 04               INR     B
034A CDFF05           CALL    COMPA
034D C24703           JNZ     FTM3
              ;
0350 41               MOV     B,C          ;INITIALIZE TEST DATA
0351 2AD783           LHLD    SVDAT        ;INITIALIZE START ADD.
0354 2B               DCX     H
0355 23       FTM4:   INX     H
0356 78               MOV     A,B
0357 BE               CMP     M            ;COMPARE EXPECTED VALUE WITH READ DATA
0358 C27B03           JNZ     ERROR        ;IF EXPECTED VALUE IS DIFFERENT FROM READ DATA THEN ERROR
035B 04       FTM5:   INR     B
035C CDFF05           CALL    COMPA
035F C25503           JNZ     FTM4
0362 0C               INR     C            ;SET NEXT TEST DATA
0363 C23F03           JNZ     FTM2         ;IF END OF TEST MEMORY THEN CONTINUE
              ;
0366 CD2906           CALL    CA           ;CLEAR (ADW) & (AR)
0369 211409           LXI     H,914H
036C 22D183           SHLD    DDW2         ;GET "GO"
036F 210D14           LXI     H,140DH
0372 22CF83           SHLD    DDW4         ;GET "OD"
0375 CDE905           CALL    FLAGC        ;RESET EACH FLAG
0378 C37C00           JMP     START
              ;
037B 22EE83   ERROR:  SHLD    ADRES        ;SET ERROR ADD.
037E 60               MOV     H,B          ;SET EXPECTED VALUE
037F 6E               MOV     L,M          ;SET READ DATA
0380 22EC83           SHLD    DATA
0383 C5               PUSH    B
0384 D5               PUSH    D
0385 E5               PUSH    H
0386 CD4E06           CALL    CH
0389 CD6C05           CALL    SEGCV
038C CD8406           CALL    KEYIN
038F E1               POP     H
0390 D1               POP     D
0391 C1               POP     B
0392 FE15             CPI     15H
0394 2AEE83           LHLD    ADRES
0397 CA5B03           JZ      FTM5         ;IF INPUT DATA IS EQUAL TO ENTER THEN REPEAT
039A 47               MOV     B,A
039B CDE905           CALL    FLAGC        ;RESET EACH FLAG
039E 78               MOV     A,B
039F C38200           JMP     STAT1
              ;
              ;********************************
              ;**   MOVE PROCESS            **
              ;********************************
              ;
```

```
03A2 3ADB83    FMOV:    LDA     ENTCT
03A5 A7                 ANA     A
03A6 C2B603             JNZ     FMOV1           :IF ENTCNT IS GREATER THAN 0 THEN FMOV1
               ;
03A9 3C                 INR     A
03AA 32DB83             STA     ENTCT           :INCREMENT ENTCNT
03AD CD7C01             CALL    ARSET           :MOVE (DR) TO (AR)
03B0 CD5106             CALL    AWCH            :CONVERT (AR) INTO (ADW)
03B3 C3E300             JMP     DTCLA
               ;
03B6 0F        FMOV1:   RRC
03B7 D2D203             JNC     FMOV2           :IF ENTCNT IS GREATER THAN 1 THEN FMOV2
03BA 3E02               MVI     A,2
03BC 32DB83             STA     ENTCT           :INCREMENT ENTCNT
03BF 2AEC83             LHLD    DATA
03C2 22D783             SHLD    SVDAT           :SAVE END ADD. OF SOURCE DATA
03C5 EB                 XCHG                    :DE : END ADD. OF SOURCE DATA
03C6 2AEE83             LHLD    ADRES           :HL : START ADD. OF SOURCE DATA
03C9 22D983             SHLD    SVDAT+2         :SAVE START ADD. OF SOURCE DATA
03CC CD1206             CALL    ERRMS           :IF START ADD. IS GREATER THAN END ADD. THEN ERROR
03CF C3E000             JMP     ALCLA
               ;
03D2 CD7C01    FMOV2:   CALL    ARSET           :MOVE (DR) TO (AR)
03D5 E5                 PUSH    H
03D6 CD5106             CALL    AWCH            :SET START ADD. OF DESTINATION DATA
03D9 CD3906             CALL    CD              :CLEAR (DDW) & (DR)
03DC CD6C05             CALL    SEGCV           :SEGMENT DATA CONVERSION
03DF 2AD783             LHLD    SVDAT
03E2 E5                 PUSH    H
03E3 C1                 POP     B               :BC : END ADD. OF SOURCE DATA
03E4 D1                 POP     D               :DE : START ADD. OF DESTINATION DATA
03E5 2AD983             LHLD    SVDAT+2         :HL : START ADD. OF SOURCE DATA
               ;
03E8 CDFF05             CALL    COMPA
03EB D20704             JNC     BTMUP           :IF HL IS LESS THAN OR EQUAL TO DE THEN EXECUTE BOTTOM UP
                                                :IF HL IS GREATER THAN DE THEN EXECUTE TOP DOWN
03EE 2B                 DCX     H
03EF 1B                 DCX     D
03F0 23        TPDWN:   INX     H
03F1 13                 INX     D
03F2 7E                 MOV     A,M
03F3 12                 STAX    D               :MOVE DATA
03F4 CD0C06             CALL    CHLBC           :IF START ADD. IS NOT EQUAL TO END ADD. THEN MOVE AGAIN
03F7 C2F003             JNZ     TPDWN
               ;
03FA EB        DSPDE:   XCHG
03FB 22EC83             SHLD    DATA            :SET END ADD. OF DESTINATION DATA
03FE CD6406             CALL    DWCH            :CONVERT (DR) INTO (DDW)
0401 CDE905             CALL    FLAGC           :RESET EACH FLAG
0404 C37C00             JMP     START
               ;
0407 C5        BTMUP:   PUSH    B               :SAVE HL (BC)
0408 C5                 PUSH    B
0409 E5                 PUSH    H
040A C1                 POP     B
040B E1                 POP     H               :EXCHANGE HL FOR BC
040C 19                 DAD     D
040D 7D                 MOV     A,L
040E 91                 SUB     C
040F 5F                 MOV     E,A
0410 7C                 MOV     A,H
0411 98                 SBB     B
0412 57                 MOV     D,A             :CALUCULATE (HL+DE-BC) AND SET ITS SOLUTION
0413 E1                 POP     H
0414 D5                 PUSH    D               :SAVE END ADD. OF DESTINATION DATA
0415 23                 INX     H
0416 13                 INX     D
0417 2B        BTUP1:   DCX     H
0418 1B                 DCX     D
0419 7E                 MOV     A,M
041A 12                 STAX    D               :MOVE DATA
041B CD0C06             CALL    CHLBC           :IF START ADD. IS NOT EQUAL TO END ADD. THEN MOVE AGAIN
041E C21704             JNZ     BTUP1
0421 D1        BTUP2:   POP     D               :RECOVER END ADD. OF DESTINATION DATA
0422 C3FA03             JMP     DSPDE
               ;
               ;*******************************
               ;**    SAVE PROCESS          **
               ;*******************************
               ;
0425 3ADB83    FSAVE:   LDA     ENTCT
0428 A7                 ANA     A
0429 C23904             JNZ     FSAV1           :IF ENTCNT IS GREATER THAN 0 THEN FSAV1
042C 3C                 INR     A
042D 32DB83             STA     ENTCT           :INCREMENT ENTCNT
0430 3AEC83             LDA     DATA
0433 32D783             STA     SVDAT           :SAVE FILE NO.
0436 C3E000             JMP     ALCLA
               ;
0439 0F        FSAV1:   RRC
043A D24B04             JNC     FSAV2           :IF ENTCNT IS GREATER THAN 1 THEN FSAV2
043D 3E02               MVI     A,2
043F 32DB83             STA     ENTCT           :INCREMENT ENTCNT
0442 CD7C01             CALL    ARSET           :SET START ADD.
0445 CD5106             CALL    AWCH            :CONVERT (AR) INTO (ADW).
0448 C3E300             JMP     DTCLA
               ;
044B 2AEC83    FSAV2:   LHLD    DATA
044E EB                 XCHG                    :DE : END ADD.
044F 2AEE83             LHLD    ADRES           :HL : START ADD.
0452 CD1206             CALL    ERRMS           :IF START ADD. IS GREATER THAN END ADD. THEN ERROR
0455 D5                 PUSH    D               :SAVE END ADD.
0456 E5                 PUSH    H               :SAVE START ADD.
```

```
0457 211D05           LXI    H,051DH
045A 22D583           SHLD   ADW2          ;GET "S-"
045D 3AD783           LDA    SVDAT
0460 CD7706           CALL   WORDC
0463 22D383           SHLD   ADW4          ;SET FILE NO.
0466 CD3906           CALL   CD            ;CLEAR (DDW) & (DR)
0469 CD6C05           CALL   SEGCV         ;SEGMENT DATA CONVERSION
                ;
046C 0E00             MVI    C,0           ;CLEAR CHECK SUM
046E CDE706           CALL   REDER         ;OUTPUT READER TO CMT
0471 BE               CMP    M             ;7 STATES DUMMY
0472 3E55             MVI    A,KEYWD
0474 0637             MVI    B,PS4         ;SET PS4 BECAUSE IT TAKES 39 STATES IN THIS INTERVAL
0476 CD0107           CALL   SRIOT         ;OUTPUT KEYWORD (END OF FILE'S HEAD)
0479 23               INX    H             ;6 STATES DUMMY
047A 2B               DCX    H             ;6 STATES DUMMY
047B BE               CMP    M             ;7 STATES DUMMY
                ;
047C 3AD783           LDA    SVDAT
047F 0635             MVI    B,PS3         ;SET PS3 BECAUSE IT TAKES 110 STATES IN THIS INTERVAL
0481 CDD204           CALL   CKSMO         ;OUTPUT FILE NO.
0484 00               NOP                  ;4 STATES DUMMY
0485 00               NOP                  ;4 STATES DUMMY
0486 E1               POP    H             ;RECOVER START ADD.
0487 D1               POP    D             ;RECOVER END ADD.
0488 7C               MOV    A,H
0489 0635             MVI    B,PS3         ;SET PS3 BECAUSE IT TAKES 110 STATES IN THIS INTERVAL
048B CDD204           CALL   CKSMO         ;OUTPUT START ADD.H
048E 7D               MOV    A,L
048F 0637             MVI    B,PS4         ;SET PS4 BECAUSE IT TAKES 82 STATES IN THIS INTERVAL
0491 CDD204           CALL   CKSMO         ;OUTPUT START ADD.L
0494 7A               MOV    A,D
0495 0637             MVI    B,PS4         ;SET PS4 BECAUSE IT TAKES 82 STATES IN THIS INTERVAL
0497 CDD204           CALL   CKSMO         ;OUTPUT END ADD.H
049A 7B               MOV    A,E
049B 0637             MVI    B,PS4         ;SET PS4 BECAUSE IT TAKES 82 STATES IN THIS INTERVAL
049D CDD204           CALL   CKSMO         ;OUTPUT END ADD.L
04A0 34               INR    M             ;10 STATES DUMMY
04A1 35               DCR    M             ;10 STATES DUMMY
04A2 79               MOV    A,C
04A3 2F               CMA
04A4 3C               INR    A             ;2'S COMPLEMENT
04A5 0635             MVI    B,PS3         ;SET PS3 BECAUSE IT TAKES 110 STATES IN THIS INTERVAL
04A7 CDD204           CALL   CKSMO         ;OUTPUT ADDRESS CHECKSUM
                ;
04AA 3AEC83           LDA    DATA          ;13 STATES DUMMY
04AD 0635             MVI    B,PS3         ;SET PS3 BECAUSE IT TAKES 110 STATES IN THIS INTERVAL
04AF 2B               DCX    H
04B0 23         FSAV3: INX   H
04B1 7E               MOV    A,M
04B2 CDD204           CALL   CKSMO         ;OUTPUT DATA
04B5 DBF8             IN     PORTA         ;10 STATES DUMMY
04B7 0631             MVI    B,PS1         ;SET PS1 BECAUSE IT TAKES 166 STATES IN THIS INTERVAL
04B9 CDFF05           CALL   COMPA
04BC C2B004           JNZ    FSAV3
04BF 00               NOP                  ;4 STATES DUMMY
04C0 BE               CMP    M             ;7 STATES DUMMY
04C1 79               MOV    A,C
04C2 2F               CMA
04C3 3C               INR    A             ;2'S COMPLEMENT
04C4 0630             MVI    B,PS2         ;SET PS2 BECAUSE IT TAKES 180 STATES IN THIS INTERVAL
04C6 CDD204           CALL   CKSMO         ;OUTPUT DATA CHECK SUM
                ;
04C9 CD3906           CALL   CD            ;CLEAR (DDW) & (DR)
04CC CDE905           CALL   FLAGC         ;RESET EACH FLAG
04CF C37C00           JMP    START
                ;
04D2 F5         CKSMO: PUSH  PSW
04D3 81               ADD    C
04D4 4F               MOV    C,A
04D5 F1               POP    PSW
04D6 32FF83           STA    DIG+7         ;DISPLAY OUTPUT DATA
04D9 C30107           JMP    SRIOT         ;OUTPUT 1 BYTE TO CMT
                ;
                ;*********************************
                ;**   LOAD PROCESS             **
                ;*********************************
                ;
04DC 3AEC83     FLOAD: LDA   DATA
04DF F5               PUSH   PSW
04E0 CD7706           CALL   WORDC
04E3 22D383           SHLD   ADW4          ;SET FILE NO.
04E6 211D05           LXI    H,051DH
04E9 22D583           SHLD   ADW2          ;GET "S-"
04EC CD3906           CALL   CD            ;CLEAR (DDW) & (DR)
                ;
04EF CD6C05     FILE:  CALL  SEGCV         ;SEGMENT DATA CONVERSION
04F2 0E00             MVI    C,0           ;CLEAR CHECK SUM
04F4 CD6507           CALL   RDSCH         ;SEARCH HEAD OF FILE
04F7 CD8307           CALL   SRIIN
04FA FE55             CPI    KEYWD
04FC C2EF04           JNZ    FILE          ;IF HEAD OF FILE WAS FOUND THEN READ FILE NO.
                ;
04FF CD6105           CALL   CKSMI         ;INPUT & DISPLAY FILE NO.
0502 57               MOV    D,A
0503 F1               POP    PSW
0504 BA               CMP    D
0505 CA1305           JZ     FOUND         ;IF FILE NO.IS DIFFERENT FROM SERCHING ONE THEN IGNORE
                ;
0508 F5               PUSH   PSW
0509 7A               MOV    A,D
050A CD7706           CALL   WORDC
050D 22CF83           SHLD   DDW4          ;DISPLAY SKIPPING FILE NO.
```

```
0510 C3EF04              JMP     FILE
                 ;
0513 AF          FOUND:  XRA     A
0514 32FE83              STA     DIG+6
0517 3E71                MVI     A,71H
0519 32F883              STA     DIG              ;DISPLAY FILE FOUND MESSAGE
051C CD6105              CALL    CKSMI
051F 67                  MOV     H,A
0520 CD6105              CALL    CKSMI
0523 6F                  MOV     L,A
0524 CD6105              CALL    CKSMI
0527 57                  MOV     D,A
0528 CD6105              CALL    CKSMI
052B 5F                  MOV     E,A
052C CD6105              CALL    CKSMI
052F C25605              JNZ     EREAD            ;IF ADDRESS READ ERROR THEN EREAD
                 ;
0532 E5                  PUSH    H                ;SAVE START ADD.
0533 2B                  DCX     H
0534 23          LOAD1:  INX     H
0535 CD6105              CALL    CKSMI            ;INPUT DATA
0538 77                  MOV     M,A              ;SET DATA TO MEMORY
0539 CDFF05              CALL    COMPA
053C C23405              JNZ     LOAD1
053F CD6105              CALL    CKSMI
0542 E1                  POP     H                ;RECOVER START ADD.
0543 C25605              JNZ     EREAD            ;IF DATA READ ERROR THEN EREAD
                 ;
0546 22EE83              SHLD    ADRES            ;SET START ADD.
0549 EB                  XCHG
054A 22EC83              SHLD    DATA             ;SET END ADD.
054D CD4E06              CALL    CH               ;CONVERT (AR),(DR) INTO (ADW),(DDW)
0550 CDE905              CALL    FLAGC            ;RESET EACH FLAG
0553 C37C00              JMP     START
                 ;
0556 3E0E        EREAD:  MVI     A,0EH
0558 32D683              STA     ADW1             ;GET "E"
055B CDE905              CALL    FLAGC            ;RESET EACH FLAG
055E C3E300              JMP     DTCLA
                 ;
0561 CD8307      CKSMI:  CALL    SRIIN
0564 32FF83              STA     DIG+7            ;DISPLAY INPUT DATA
0567 47                  MOV     B,A
0568 81                  ADD     C
0569 4F                  MOV     C,A
056A 78                  MOV     A,B
056B C9                  RET
                 ;
                 ;*******************************
                 ;**  SEGMENT DATA CONVERSION **
                 ;*******************************
                 ;
056C 21D683      SEGEV:  LXI     H,ADW1           ;SET ADD.& DATA DISPLAY REGISTER ADD.
056F 11F883              LXI     D,DIG            ;SET SEG BUFF ADD.
0572 01B305              LXI     B,SEGDA          ;SET SEG DATA ADD.
0575 3ADF83              LDA     FMODE            ;GET MODE FLAG
0578 E63C                ANI     3CH              ;SET MASK OF MODE FLAG
057A 07                  RLC
057B 07                  RLC
057C F5                  PUSH    PSW
057D 7E          SEGRP:  MOV     A,M
057E E5                  PUSH    H
057F 2600                MVI     H,0
0581 6F                  MOV     L,A
0582 09                  DAD     B
0583 7E                  MOV     A,M              ;GET SEG DATA
0584 12                  STAX    D                ;MOVE SEG DATA TO SEG BUFF
0585 E1                  POP     H
0586 F1                  POP     PSW
0587 07                  RLC
0588 F5                  PUSH    PSW
0589 DCAE05              CC      PRIOD            ;IF EACH MODE FLAG IS ON THEN DISPLAY 1 DOT ON (DIG--DIG+3)
058C 2B                  DCX     H
058D 1C                  INR     E
058E C27D05              JNZ     SEGRP
0591 F1                  POP     PSW              ;DUMMY

0592 3ADF83              LDA     FMODE            ;GET MODE FLAG
0595 0F                  RRC
0596 D2A605              JNC     REGPI            ;IF F-MODE IS ON THEN DISPLAY 4 DOTS ON (DIG--DIG+3)
0599 0604                MVI     B,4
059B 11F883              LXI     D,DIG
059E CDAE05      SEGMD:  CALL    PRIOD
05A1 13                  INX     D
05A2 05                  DCR     B
05A3 C29E05              JNZ     SEGMD
                 ;
05A6 3ADE83      REGPI:  LDA     FREG             ;GET REGISTER FLAG
05A9 0F                  RRC
05AA D0                  RNC
05AB 11FC83              LXI     D,DIG+4          ;IF REGISTER FLAG IS ON THEN DISPLAY 1 DOT ON (DIG+4)
                 ;
05AE 1A          PRIOD:  LDAX    D
05AF F680                ORI     80H
05B1 12                  STAX    D
05B2 C9                  RET
                 ;
05B3 3F065B      SEGDA:  DB      3FH,06H,5BH,4FH,66H,6DH,7DH,27H
05B6 4F666D
05B9 7D27
05BB 7F6F77              DB      7FH,6FH,77H,7CH,39H,5EH,79H,71H
05BE 7C395E
05C1 7971
```

```
05C3 761E38          DB      76H,1EH,38H,54H,5CH,73H,67H,50H
05C6 545C73
05C9 6750
05CB 3E1C6E          DB      3EH,1CH,6EH,74H,00H,40H,80H
05CE 740040
05D1 80
                ;
                ;******************************
                ;** SHIFT (DR),(DDW)        **
                ;******************************
                ;
05D2 2AD083     SHIFT:  LHLD    DDW3
05D5 22D183             SHLD    DDW2
05D8 3ACF83             LDA     DDW4
05DB 32D083             STA     DDW3
05DE 2AEC83             LHLD    DATA
05E1 29                 DAD     H
05E2 29                 DAD     H
05E3 29                 DAD     H
05E4 29                 DAD     H
05E5 22EC83             SHLD    DATA
05E8 C9                 RET
                ;
                ;******************************
                ;**  RESET EACH FLAG        **
                ;******************************
                ;
05E9 210000     FLAGC:  LXI     H,0
05EC 22DB83             SHLD    ENTCT              ;CLEAR ENTER COUNTER & RESET TRAP FLAG
05EF 22DE83             SHLD    FREG               ;RESET REGISTER & MODE FLAG
05F2 3E01               MVI     A,1
05F4 C9                 RET
                ;
                ;******************************
                ;**    SET BIT              **
                ;******************************
                ;
05F5 3C         STBIT:  INR     A
05F6 4F                 MOV     C,A                ;SET COUNTER
05F7 3E80               MVI     A,80H
05F9 07         LSIFT:  RLC                        ;SHIFT LEFT 1 BIT
05FA 0D                 DCR     C
05FB C2F905             JNZ     LSIFT
05FE C9                 RET
                ;
                ;
                ;
                ;********************************
                ;**    COMPARE(HL:DE,HL:BC)   **
                ;********************************
                ;
05FF 7A         COMPA:  MOV     A,D
0600 BC                 CMP     H
0601 C20B06             JNZ     NTEQU
0604 00                 NOP                        ;4 STATES DUMMY
0605 7B                 MOV     A,E
0606 BD                 CMP     L
0607 C9                 RET
0608 CC0000     NTEQU:  CZ      0                  ;9 STATES DUMMY
060B C9                 RET                        ;IF HL IS EQUAL TO DE THEN SET ZERO FLAG
                                                   ;IF HL IS NOT EQUAL TO DE THEN RESET ZERO FLAG
                                                   ;IF HL IS GREATER THAN DE THEN SET CARRY FLAG
                                                   ;IF HL IS LESS THAN OR EQUAL TO DE THEN RESET CARRY FLAG
                                                   ;IT TAKES 37 STATES IN THIS SUBROUTINE
                ;
060C 78         CHLBC:  MOV     A,B
060D BC                 CMP     H
060E C0                 RNZ
060F 79                 MOV     A,C
0610 BD                 CMP     L
0611 C9                 RET                        ;IF HL IS EQUAL TO BC THEN SET ZERO FLAG
                                                   ;IF HL IS NOT EQUAL TO BC THEN RESET ZERO FLAG
                ;
0612 CDFF05     ERRMS:  CALL    COMPA
0615 D0                 RNC
0616 E1                 POP     H                  ;DUMMY
0617 210E1C     ERR:    LXI     H,1C0EH
061A 22D583             SHLD    ADW2               ;GET ' E'
061D 211717             LXI     H,1717H
0620 22D383             SHLD    ADW4               ;GET 'RR'
0623 CDE905             CALL    FLAGC              ;RESET EACH FLAG
0626 C3E300             JMP     DTCLA
                                                   ;IF HL IS GREATER THAN DE THEN DISPLAY ERROR MESSAGE
                                                   ;         ELSE RETURN AND CONTINUE
                ;
                ;******************************
                ;**    CLEAR DISP REG       **
                ;******************************
                ;
0629 211C1C     CA:     LXI     H,1C1CH
062C 22D583             SHLD    ADW2
062F 22D383             SHLD    ADW4               ;PUT OUT ADD.DISPLAY
0632 210000             LXI     H,0
0635 22EE83             SHLD    ADRES              ;SET 0 IN (AR)
0638 C9                 RET
                ;
0639 211C1C     CD:     LXI     H,1C1CH
063C 22CF83             SHLD    DDW4               ;PUT OUT L.DATA DISPLAY
063F AF                 XRA     A
0640 32EC83             STA     DATA               ;SET 0 IN (DR)L
0643 211C1C     CD12:   LXI     H,1C1CH
0646 22D183             SHLD    DDW2               ;PUT OUT H.DATA DISPLAY
```

```
0649 AF                 XRA     A
064A 32ED83             STA     DATA+1          ;SET 0 IN (DR)H
064D C9                 RET
                ;
                ;********************************
                ;** (ADW),(DDW)<--(AR),(DR)    **
                ;********************************
                ;
064E CD6406     CH:     CALL    DWCH
                                                ;CONVERT (AR),(DR) INTO (ADW),(DDW)
0651 3AEF83     AWCH:   LDA     ADRES+1
0654 CD7706             CALL    WORDC
0657 22D583             SHLD    ADW2            ;CONVERT (AR)H INTO (ADW1,2)
065A 3AEE83             LDA     ADRES
065D CD7706             CALL    WORDC
0660 22D383             SHLD    ADW4            ;CONVERT (AR)L INTO (ADW3,4)
0663 C9                 RET
                ;
0664 3AED83     DWCH:   LDA     DATA+1
0667 CD7706             CALL    WORDC
066A 22D183             SHLD    DDW2            ;CONVERT (DR)H INTO (DDW1,2)
066D 3AEC83     DLWCH:  LDA     DATA
0670 CD7706             CALL    WORDC
0673 22CF83             SHLD    DDW4            ;CONVERT (DR)L INTO (DDW3,4)
0676 C9                 RET
                ;
0677 47         WORDC:  MOV     B,A
0678 E60F               ANI     0FH
067A 6F                 MOV     L,A
067B 78                 MOV     A,B
067C 0F                 RRC
067D 0F                 RRC
067E 0F                 RRC
067F 0F                 RRC
0680 E60F               ANI     0FH
0682 67                 MOV     H,A
0683 C9                 RET                     ;CONVERT A INTO HL
                ;
                ;*****************************
                ;**   KEY IN                **
                ;*****************************
                ;
0684 CD9006     KEYIN:  CALL    INPUT
0687 3ADD83             LDA     FKEY            ;GET KEY FLAG
068A A7                 ANA     A
068B CA8406             JZ      KEYIN           ;IF THERE IS NOT KEY INPUT THEN KEYIN
068E 78                 MOV     A,B
068F C9                 RET
                ;
                ;
               .;
                ;*****************************
                ;**   KEY INPUT             **
                ;*****************************
                ;
0690 CDBC06     INPUT:  CALL    KEY
0693 3C                 INR     A
0694 CAB706             JZ      NOKEY           ;IF THERE IS NOT KEY INPUT THEN NOKEY
0697 160E       INPRP:  MVI     D,MSEC9         ;SET DELAY TIMER
0699 1E00               MVI     E,0
069B CDBD07             CALL    DELEN           ;WAIT CHATTERRING TIME
069E 15                 DCR     D
069F C29906             JNZ     INPRP+2
06A2 CDBC06             CALL    KEY
06A5 47                 MOV     B,A
06A6 3C                 INR     A
06A7 CAB706             JZ      NOKEY           ;IF THERE IS NOT KEY INPUT THEN NOKEY
06AA 3ADD83             LDA     FKEY            ;GET KEY FLAG
06AD A7                 ANA     A
06AE C29706             JNZ     INPRP           ;IF KEY FLAG IS EQUAL TO FF THEN INPRP
06B1 3D                 DCR     A               ;SET FF
06B2 32DD83     SETKF:  STA     FKEY            ;SET KEY FLAG
06B5 78                 MOV     A,B             ;SET INPUT DATA
06B6 C9                 RET
06B7 06FF       NOKEY:  MVI     B,0FFH
06B9 C3B206             JMP     SETKF
                ;
                ;*****************************
                ;**   KEY SCAN              **
                ;*****************************
                ;
06BC AF         KEY:    XRA     A               ;CLEAR A
06BD 57                 MOV     D,A             ;CLEAR D
06BE 47                 MOV     B,A             ;CLEAR B
06BF 3EEF               MVI     A,PC4
06C1 CDD406             CALL    SCAN
06C4 0608               MVI     B,8
06C6 3EDF               MVI     A,PC5
06C8 CDD406             CALL    SCAN
06CB 0610               MVI     B,10H
06CD 3EBF               MVI     A,PC6
06CF CDD406             CALL    SCAN
06D2 3D                 DCR     A               ;SET FF
06D3 C9                 RET
06D4 D3FA       SCAN:   OUT     PORTC           ;SCAN EACH BIT OF PORT C
06D6 DBF8               IN      PORTA           ;INPUT DATA FROM PORT A
06D8 2F                 CMA
06D9 A7                 ANA     A
06DA C8                 RZ
06DB E1                 POP     H               ;DUMMY
06DC 0F         KEY1:   RRC
```

```
06DD DAE406            JC      KEY2
06E0 14                INR     D
06E1 C3DC06            JMP     KEY1
06E4 7A        KEY2:   MOV     A,D
06E5 B0                ORA     B
06E6 C9                RET
               ;
               ;********************************
               ;**   OUTPUT READER FOR 5SEC **
               ;********************************
               ;
06E7 C5        REDER:  PUSH    B               ;SAVE BC
06E8 E5                PUSH    H               ;SAVE HL
06E9 217017            LXI     H,SEC5          ;SET LOOP COUNTER
               ;
06EC CD3B07    REDR1:  CALL    ONEOT           ;OUTPUT '1'
06EF 46                MOV     B,M             ;7 STATES DUMMY
06F0 061C              MVI     B,PRA           ;SET DELAY COUNTER
06F2 2D                DCR     L
06F3 C2EC06            JNZ     REDR1
               ;
06F6 DBF8              IN      PORTA           ;10 STATES DUMMY
06F8 061A              MVI     B,PRB           ;SET DELAY COUNTER
06FA 25                DCR     H
06FB C2EC06            JNZ     REDR1
               ;
06FE E1                POP     H               ;RECOVER HL
06FF C1                POP     B               ;RECOVER BC
0700 C9                RET
               ;
               ;********************************
               ;**   OUTPUT ONE BYTE          **
               ;********************************
               ;
0701 C5        SRIOT:  PUSH    B               ;SAVE BC
0702 D5                PUSH    D               ;SAVE DE
0703 E5                PUSH    H               ;SAVE HL
               ;
0704 4F                MOV     C,A             ;SAVE 1 BYTE DATA
0705 CD4A07            CALL    ZEROT           ;WRITE START BIT (0)
0708 113E19            LXI     D,P10           ;SET DELAY COUNTER
070B 7E                MOV     A,M             ;7 STATES DUMMY
070C 2E08              MVI     L,8             ;SET LOOP COUNTER
               ;
070E 79        SROT1:  MOV     A,C             ;RECOVER 1 BYTE DATA
070F 1F                RAR
0710 4F                MOV     C,A             ;SAVE 1 BYTE DATA
0711 DA1E07            JC      SROT2           ;IF DATA IS EQUAL TO '1' THEN JUMP
               ;
0714 43                MOV     B,E             ;SET DELAY COUNTER
0715 DBF8              IN      PORTA           ;10 STATES DUMMY
0717 7E                MOV     A,M             ;7 STATES DUMMY
0718 CD4A07            CALL    ZEROT           ;WRITE 1 BIT (0)
071B C32807            JMP     SROT3
               ;
071E 42        SROT2:  MOV     B,D             ;SET DELAY COUNTER
071F DBF8              IN      PORTA           ;10 STATES DUMMY
0721 DBF8              IN      PORTA           ;10 STATES DUMMY
0723 CD3B07            CALL    ONEOT           ;WRITE 1 BIT (1)
0726 DBF8              IN      PORTA           ;10 STATES DUMMY
               ;
0728 2D        SROT3:  DCR     L
0729 C20E07            JNZ     SROT1           ;IF END OF 8 BITS THEN CONTINUE
               ;
072C 061C              MVI     B,POA           ;SET DELAY COUNTER
072E CD3B07            CALL    ONEOT           ;WRITE END BIT (1)
0731 7E                MOV     A,M             ;7 STATES DUMMY
0732 061D              MVI     B,POB           ;SET DELAY COUNTER
0734 CD3B07            CALL    ONEOT           ;WRITE END BIT (1)
               ;
0737 E1                POP     H               ;RECOVER HL
0738 D1                POP     D               ;RECOVER DE
0739 C1                POP     B               ;RECOVER BC
073A C9                RET
               ;
               ;********************************
               ;**   OUTPUT ONE BIT           **
               ;********************************
               ;
073B 3ED8      ONEOT:  MVI     A,HIGH          ;SET WAVE TO HIGH
073D CD5907            CALL    WVCG1           ;CHANGE WAVE FROM HIGH TO LOW
0740 CD5707            CALL    WVCHG           ;CHANGE WAVE FROM LOW TO HIGH
0743 CD5707            CALL    WVCHG           ;CHANGE WAVE FROM HIGH TO LOW
0746 CD5707            CALL    WVCHG           ;CHANGE WAVE FROM LOW TO HIGH
0749 C9                RET
               ;
074A 3ED8      ZEROT:  MVI     A,HIGH          ;SET WAVE TO HIGH
074C CD5907            CALL    WVCG1           ;CHANGE WAVE FROM HIGH TO LOW
074F 00                NOP                     ;4 STATES DUMMY
0750 00                NOP                     ;4 STATES DUMMY
0751 0644              MVI     B,P00           ;SET DELAY COUNTER
0753 CD5907            CALL    WVCG1           ;CHANGE WAVE FROM LOW TO HIGH
0756 C9                RET
               ;
0757 0620      WVCHG:  MVI     B,P01           ;SET DELAY COUNTER
0759 05        WVCG1:  DCR     B
075A C25907            JNZ     WVCG1           ;DELAY HALF CYCLE
075D 03                INX     B               ;6 STATES DUMMY
075E 0B                DCX     B               ;6 STATES DUMMY
075F 00                NOP                     ;4 STATES DUMMY
```

```
0760 17             RAL
0761 3F             CMC                    ;CHANGE WAVE DATA
0762 1F             RAR
0763 30             SIM                    ;OUTPUT SERIAL DATA
0764 C9             RET
                ;
                ;*****************************
                ;**   READER SEARCH         **
                ;*****************************
                ;
0765 C5         RDSCH:  PUSH  B            ;SAVE BC
0766 D5                 PUSH  D            ;SAVE DE
                ;
0767 20                 RIM                ;SERIAL INPUT
0768 47                 MOV   B,A          ;SAVE INPUT DATA
0769 1600       RDSC1:  MVI   D,0          ;CLEAR 256 COUNTER
076B 1E00       RDSC2:  MVI   E,0          ;CLEAR DELAY COUNTER
076D CDC607             CALL  SERCH        ;SEARCH WAVE CHANGE
0770 3E1F               MVI   A,B1MAX      ;SET DELAY COUNTER TO MAX. THRESHOLD OF DATA '1'
0772 BB                 CMP   E
0773 DA6907             JC    RDSC1        ;IF NOT READER THEN SEARCH READER AGAIN
0776 3E08               MVI   A,B1MIN      ;SET DELAY COUNTER TO MIN. THRESHOLD OF DATA '1'
0778 BB                 CMP   E
0779 D26907             JNC   RDSC1        ;IF NOT READER THEN SEARCH READER AGAIN
077C 14                 INR   D
077D C26B07             JNZ   RDSC2        ;IF FREQUENCY OF 256 WAVES IS EQUAL TO 2400HZ THEN CONTINUE
                ;
0780 D1                 POP   D            ;RECOVER DE
0781 C1                 POP   B            ;RECOVER BC
0782 C9                 RET
                ;
                ;*****************************
                ;**   INPUT ONE BYTE        **
                ;*****************************
                ;
0783 C5         SRIIN:  PUSH  B            ;SAVE BC
0784 D5                 PUSH  D            ;SAVE DE
0785 E5                 PUSH  H            ;SAVE HL
                ;
0786 20                 RIM                ;SERIAL INPUT
0787 47                 MOV   B,A          ;SAVE INPUT DATA
0788 1E00       SRII1:  MVI   E,0          ;CLEAR DELAY COUNTER
078A CDC607             CALL  SERCH        ;SEARCH WAVE CHANGE
078D 3E20               MVI   A,B0FWD      ;SET DELAY COUNTER TO THRESHOLD OF DATA '0'
078F BB                 CMP   E
0790 D28807             JNC   SRII1        ;IF DATA IS EQUAL TO 1 THEN SEARCH START BIT AGAIN
                ;
0793 3E4A               MVI   A,B0BCK      ;SET DELAY COUNTER
0795 CDBF07             CALL  DELE1        ;DELAY
0798 2E08               MVI   L,8          ;SET 8BITS LOOP COUNTER
079A CDAC07     SRII2:  CALL  ONEIN        ;INPUT ONE BIT DATA
079D 7C                 MOV   A,H          ;LOAD ONE BYTE DATA
079E 1F                 RAR
079F 67                 MOV   H,A          ;STORE ONE BYTE DATA
07A0 CDBD07             CALL  DELEN        ;DELAY
07A3 2D                 DCR   L
07A4 C29A07             JNZ   SRII2        ;IF END OF 8 BITS THEN CONTINUE
07A7 7C                 MOV   A,H          ;SET ONE BYTE DATA TO A
                ;
07A8 E1                 POP   H            ;RECOVER HL
07A9 D1                 POP   D            ;RECOVER DE
07AA C1                 POP   B            ;RECOVER BC
07AB C9                 RET
                ;
                ;*****************************
                ;**   INPUT ONE BIT         **
                ;*****************************
                ;
07AC CDC607     ONEIN:  CALL  SERCH        ;SEARCH WAVE CHANGE
07AF 3E35       ONEI1:  MVI   A,C58TH      ;SET DELAY COUNTER
07B1 1E00               MVI   E,0          ;CLEAR DELAY COUNTER
07B3 CDBF07             CALL  DELE1        ;DELAY FOR 5/8 CYCLE
07B6 CDC607             CALL  SERCH        ;SEARCH WAVE CHANGE
07B9 3E37               MVI   A,C58TH+2    ;LOAD DATA THRESHOLD
07BB BB                 CMP   E
07BC C9                 RET                ;IF ONE BIT DATA IS EQUAL TO '1' THEN SET CARRY
                                           ;IF ONE BIT DATA IS EQUAL TO '0' THEN RESET CARRY
                ;
                ;*****************************
                ;**   DELAY TIMER           **
                ;*****************************
                ;
07BD 3E47       DELEN:  MVI   A,DELAY      ;SET DELAY COUNTER
07BF 1C         DELE1:  INR   E
07C0 00                 NOP                ;4 STATES DUMMY
07C1 BB                 CMP   E
07C2 D2BF07             JNC   DELE1
07C5 C9                 RET
                ;
                ;*****************************
                ;**   SEARCH WAVE CHANGE    **
                ;*****************************
                ;
07C6 1C         SERCH:  INR   E
07C7 20                 RIM                ;SERIAL INPUT
07C8 A8                 XRA   B
07C9 CAC607             JZ    SERCH        ;WAVE NOT CHANGE THEN SEARCH AGAIN
07CC A8                 XRA   B
07CD 47                 MOV   B,A          ;SAVE INPUT DATA
```

```
07CE C9                RET
                 ;
                 ;********************************
                 ;**   DISPLAY (AR),(DR)       **
                 ;********************************
                 ;
07CF 21EF83      RGDSP:  LXI     H,ADRES+1
07D2 11F483              LXI     D,DISP
07D5 0604                MVI     B,4
07D7 7E          RGDS1:  MOV     A,M
07D8 12                  STAX    D
07D9 2B                  DCX     H
07DA 13                  INX     D
07DB 05                  DCR     B
07DC C2D707              JNZ     RGDS1
                 ;
                 ;********************************
                 ;**  SEGMENT DATA CONVERSION  **
                 ;********************************
                 ;
07DF 0604        SEGCG:  MVI     B,4
07E1 21F483              LXI     H,DISP
07E4 11D683              LXI     D,ADW1
07E7 7E          SEGC1:  MOV     A,M
07E8 CDF707              CALL    SEGC2
07EB 7E                  MOV     A,M
07EC CDFB07              CALL    SEGC3
07EF 23                  INX     H
07F0 05                  DCR     B
07F1 C2E707              JNZ     SEGC1
07F4 C36C05              JMP     SEGCV
07F7 0F          SEGC2:  RRC
07F8 0F                  RRC
07F9 0F                  RRC
07FA 0F                  RRC
07FB E60F        SEGC3:  ANI     0FH
07FD 12                  STAX    D
07FE 1B                  DCX     D
07FF C9                  RET
                 ;
                 ;********************************
                 ;**    WORKING AREA           **
                 ;********************************
                 ;
0800                     ORG     8391H
8391             USESP:  DS      20H
83B1             RST1:   DS      3
83B4             RST2:   DS      3
83B7             RST3:   DS      3
83BA             RST4:   DS      3
83BD             RST5:   DS      3
83C0             RST55:  DS      3
83C3             RST6:   DS      3
83C6             RST7:   DS      3
83C9             GOIN:   DS      1
83CA             PORTI:  DS      2
83CC             GOOUT:  DS      1
83CD             PORTO:  DS      2
83CF             DDW4:   DS      1
83D0             DDW3:   DS      1
83D1             DDW2:   DS      1
83D2             DDW1:   DS      1
83D3             ADW4:   DS      1
83D4             ADW3:   DS      1
83D5             ADW2:   DS      1
83D6             ADW1:   DS      1
83D7             SVDAT:  DS      4
83DB             ENTCT:  DS      1
83DC             FTRAP:  DS      1          ;(00H:SYSTEM FFH:USER)
83DD             FKEY:   DS      1          ;(00H:NONKEY FFH:KEY )
83DE             FREG:   DS      1          ;(0:REG 1:AF 2:BC 3:DE 4:HL 5:SP 6:BP 7:BD
83DF             FMODE:  DS      1          ;(0:MOD 1:?? 2:SV 3:LD 4:TM 5:MV 6:OT 7:IN)
83E0             SVPC:   DS      2
83E2             SVSP:   DS      2
83E4             SVHL:   DS      2
83E6             SVDE:   DS      2
83E8             SVBC:   DS      2
83EA             SVAF:   DS      2
83EC             DATA:   DS      2
83EE             ADRES:  DS      2
83F0             SVBP:   DS      2
83F2             SVBD:   DS      2
83F4             DISP:   DS      4
83F8             DIG:    DS      8
                 ;
                 ;******************************
                 ;**    EQU TABLE            **
                 ;******************************
                 ;
0092             CTRLW   EQU     92H
0098             MSKRS   EQU     98H
0055             KEYWD   EQU     55H
00FB             MODES   EQU     0FBH
00FA             PORTC   EQU     0FAH
00F8             PORTA   EQU     0F8H
00EF             PC4     EQU     0EFH
00DF             PC5     EQU     0DFH
00BF             PC6     EQU     0BFH
000E             MSEC9   EQU     14
```

```
1770          SEC5    EQU    1770H
0031          PS1     EQU    49
0030          PS2     EQU    48
0035          PS3     EQU    53
0037          PS4     EQU    55
001C          PRA     EQU    28
001A          PRB     EQU    26
193E          P10     EQU    193EH
001C          POA     EQU    28
001D          POB     EQU    29
0044          PO0     EQU    68
0020          PO1     EQU    32
00D8          HIGH    EQU    0D8H
001F          B1MAX   EQU    31
0008          B1MIN   EQU    8
0020          BOFWD   EQU    32
004A          BOBCK   EQU    74
0035          C58TH   EQU    53
0047          DELAY   EQU    71
F125          BSBRK   EQU    0F125H
0000                  END
```

SYMBOL	ADRS	SYMBOL	ADRS	SYMBOL	ADRS	SYMBOL	ADRS	SYMBOL	ADRS
ADRES	83EE	ADRST	0166	ADW1	83D6	ADW2	83D5	ADW3	83D4
ADW4	83D3	ALCLA	00E0	ARSET	017C	AWCH	0651	BOBCK	004A
BOFWD	0020	B1MAX	001F	B1MIN	0008	BRDPT	026F	BRET	025D
BSBRK	F125	BTMUP	0407	BTUP1	0417	BTUP2	0421	C58TH	0035
CA	0629	CD	0639	CD12	0643	CH	064E	CHLBC	060C
CKSMI	0561	CKSMO	04D2	CLEA1	0042	CLEA2	0043	CLEAR	003F
COMPA	05FF	CONT	01A4	CONT1	01C3	CTRLW	0092	DATA	83EC
DDW1	83D2	DDW2	83D1	DDW3	83D0	DDW4	83CF	DELAY	0047
DELE1	07BF	DELEN	07BD	DIG	83F8	DIGIT	00BB	DISP	83F4
DLWCH	066D	DSPDE	03FA	DTCLA	00E3	DTSET	02A8	DWCH	0664
ENTCT	83DB	EREAD	0556	ERR	0617	ERRMS	0612	ERROR	037B
FILE	04EF	FIN	02D6	FKEY	83DD	FLAGC	05E9	FLOAD	04DC
FMODE	83DF	FMOV	03A2	FMOV1	03B6	FMOV2	03D2	FOUND	0513
FOUT	02FF	FREG	83DE	FSAV1	0439	FSAV2	044B	FSAV3	04B0
FSAVE	0425	FTM	0322	FTM1	0339	FTM2	033F	FTM3	0347
FTM4	0355	FTM5	035B	FTRAP	83DC	GOIN	83C9	GOOUT	83CC
HIGH	00D8	INIT	0202	INPRP	0697	INPUT	0690	KEY	06BC
KEY1	06DC	KEY2	06E4	KEYIN	0684	KEYWD	0055	LOAD1	0534
LSIFT	05F9	MD	00F2	MDFKP	00DA	MEMRD	0169	MODES	00FB
MSEC9	000E	MSKRS	009B	NOKEY	06B7	NTEQU	0608	ONEI1	07AF
ONEIN	07AC	ONEOT	073B	OUTIN	02F9	P10	193E	PC4	00EF
PC5	00DF	PC6	00BF	PIN	02ED	PD0	0044	PD1	0020
POA	001C	POB	001D	PORTA	00F8	PORTC	00FA	PORTI	83CA
PORTO	83CD	POUT	0319	PRA	001C	PRB	001A	PRIOD	05AE
PRMOD	02BF	PRREG	0295	PS1	0031	PS2	0030	PS3	0035
PS4	0037	RDDEC	018D	RDINC	0183	RDSC1	0769	RDSC2	076B
RDSCH	0765	REDER	06E7	REDR1	06EC	REG	00E9	REGFN	0112
REGPI	05A6	RESE1	0062	RESE2	006A	RESET	005F	RGDS1	07D7
RGDSP	07CF	RSIFT	029E	RST1	83B1	RST2	83B4	RST3	83B7
RST4	83BA	RST5	83BD	RST55	83C0	RST6	83C3	RST7	83C6
RUN	0197	SCAN	06D4	SEC5	1770	SEGC1	07E7	SEGC2	07F7
SEGC3	07FB	SEGCG	07DF	SEGCV	056C	SEGDA	05B3	SEGMD	059E
SEGRP	057D	SERCH	07C6	SETKF	06B2	SHIFT	05D2	SRII1	0788
SRII2	079A	SRIIN	0783	SRIOT	0701	SROT1	070E	SROT2	071E
SROT3	0728	START	007C	STAT1	0082	STAT2	0095	STAT3	009B
STBIT	05F5	STEP	0219	STEP1	0226	SVAF	83EA	SVBC	83E8
SVBD	83F2	SVBP	83F0	SVDAT	83D7	SVDE	83E6	SVHL	83E4
SVPC	83E0	SVSP	83E2	TFUNC	00AB	TPDWN	03F0	TRAP	01D3
TRAP1	01E4	TRGAD	0158	TRGNM	014A	USESP	8391	WORDC	0677
WTENT	0279	WVCG1	0759	WVCHG	0757	ZEROT	074A		

10章　モニタサブルーチンの使い方

1．はじめに

通常プログラムは，メインプログラムとそれに付随するいくつかのサブルーチンから構成されています．

サブルーチンプログラムとは，繰り返し行われる処理や共通に使われる処理を一連のプログラムとしてまとめたものです．

モニタプログラムにもいくつかのサブルーチンが存在しています．ここに紹介するサブルーチンは，これらの中でも一般性があり，ユーザーがアプリケーションプログラムを作成する上で，有効であると思われるものです．

TK-80が発売されて以来，これまでに数多くのアプリケーションプログラムが作成されてきました．これらのアプリケーションプログラムがそのまま使えるようにTK-80で公開したRGDSP, SEGCG, INPUT そして KEYIN という四つのサブルーチンをソフトウェアコンパチブルにしています．

2．サブルーチンの考え方

サブルーチンを使用する際に，理解しておかなければならないことがいくつかあります．

その中の一つにスタックの考え方があります．サブルーチンをコールしたとき，サブルーチン処理が終了した場合の戻り番地を記憶しておかなければなりません．この戻り番地はサブルーチンコールされたときに，スタックポインタが指している RAM 上のスタックエリアに自動的に書き込まれ，サブルーチンの終りで RET 命令を実行することによって引用されます．

したがって，一般にサブルーチンを用いるときは，あらかじめスタックポインタの設定を行う必要があります．ところが，TK-85 では，ユーザープログラムを実行させる際にモニタがスタックポインタを 8391H 番地にセットしています．したがって特別にスタックポインタの設定を行わなくても，サブルーチンコール等のスタック操作も可能なわけです．

次に考慮すべきことは，パラメータが必要なサブルーチンであるかどうかということです．サブルーチンは一連の処理を行いますが，処理によってはパラメータが必要となる場合があります．通常ライブラリーとして用意されているサブルーチンではパラメータが当然必要となります．高級言語の場合は，これらパラメータの受け渡しを引き数あるいは共通領域の使用等によって行いますが，機械語を使用する場合は，レジスタ渡し，あるいは RAM を利用して行うことになりま

す．モニタサブルーチンを使用する際は，パラメータが必要な場合には適正な設定を行い，また出力パラメータがある場合は，それをこわすことのないよう注意して下さい．

つぎに考慮すべきことは，モニタサブルーチンの処理内部で破壊されるレジスタがあるということです．したがってこわされては困るレジスタがある場合は，スタックに退避する等の対策を講じなければなりません．

最後にスタックの動作を図10．1によって具体的に説明します．

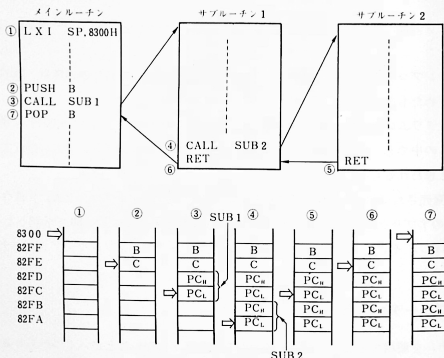


図10.1

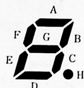
ここでちょっとしたテクニックを紹介しましょう．サブルーチン1の内部をみると，サブルーチン2へのコール命令とRET命令が続いています．このような場合上記2命令をJMP SUB2と置き換えても同じ動作を行います．JMP SUB2と置き換えた場合命令バイト数が1バイト減り，スタックレベルも1レベル少なくて済みます．

3．表示用サブルーチン

TK-85は，モニタワーキングエリア内のセグメントデータバッファ（83F8H番地から83FFH番地）の内容をDMA転送によって常時表示しています．

実際にデータをLEDディスプレイに表示させるためには，表示させるデータ（セグメントデータ）を直接上記のエリアに格納すればよいわけです．

なお一つの文字は，1バイト（8ビット）のデータで表示されますが，このデータは各ビットが次のように各セグメントに対応して構成されています．



A：BIT 0	E：BIT 4
B：BIT 1	F：BIT 5
C：BIT 2	G：BIT 6
D：BIT 3	H：BIT 7

セグメントは点灯させるセグメントに対応するビットを“1”とし，点灯させないセグメントに対応するビットを“0”として構成します．

このように，セグメントデータを直接セグメントバッファに格納すればデータは表示されますが，これでは表示させたいデータを基にセグメントデータをそのたびに計算しなければなりません．またプログラム作成の上でもかなり手間のかかることが予想されます．そこでユーザーがプログラムを作成する上で利用できるサブルーチンを公開し，表示処理による手間を省いています．

3．1　表示用サブルーチン構成

モニタには，モニタ処理に便利なように幾つかのサブルーチンが用意してありますが，それぞれのサブルーチンがどのような機能を持っているのか整理する意味で，まず図10．2に示す表示用サブルーチン構成図を理解して下さい．

3．2　機能説明

3．2．1　SEGCV

(1)　スタート番地　　056CH 番地

(2)　入出力条件　　入力パラメータ　　モードフラグ(83DFH 番地)，
レジスタフラグ(83DEH 番地)

出力パラメータ　　なし

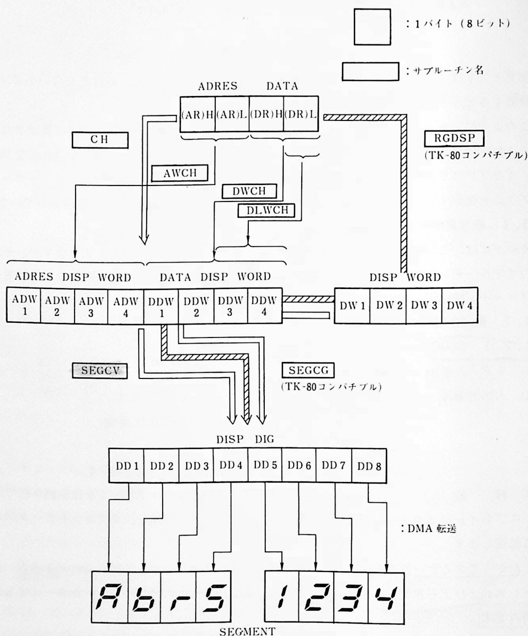
使用レジスタ　　A，F，B，C，D，E，H，L

(3)　機　　能　　アドレスおよびデータディスプレイレジスタ8バイトに格納されているディスプレイレジスタデータを LED の8セグメントデータに変換し，セグメントデータバッファに転送します．

したがってあるデータを表示させたい場合には，表示させたいディスプレイレジスタデータをアドレスおよびデータディスプレイエリアにセットしてからこのサブルーチンをコールするだけで済みます．

またユーザーに直接関係はありませんが，モードフラグおよびレジスタフラグの内容をみて，ドットを点灯させる処理を行っています．

各レジスタおよび LED ディスプレイにおけるデータ転送の関係，そしてディスプレイデータ，ディスプレイレジスタデータ，セグメントデータの関係は図10．3のようになっています．



：1バイト（8ビット）
：サブルーチン名
ADRES
DATA
(AR)H
(AR)L
(DR)H
(DR)L
CH
AWCH
DWCH
DLWCH
RGDSP
(TK-80コンパチブル)
ADRES DISP WORD
DATA DISP WORD
DISP WORD
ADW 1
ADW 2
ADW 3
ADW 4
DDW 1
DDW 2
DDW 3
DDW 4
DW 1
DW 2
DW 3
DW 4
SEGCV
SEGCG
(TK-80コンパチブル)
DISP DIG
DD 1
DD 2
DD 3
DD 4
DD 5
DD 6
DD 7
DD 8
：DMA 転送
SEGMENT

図10.2

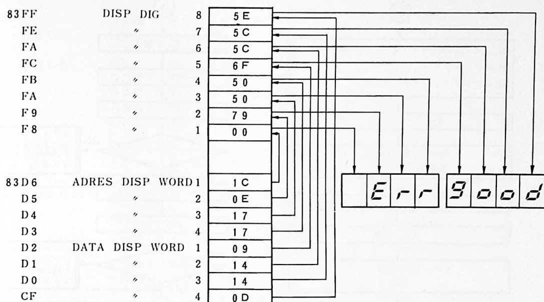


図10.3

ディスプレイデータ	0	1	2	3	4	5	6	7
ディスプレイレジスタデータ	00	01	02	03	04	05	06	07
セグメントデータ	3F	06	5B	4F	66	6D	7D	27
ディスプレイデータ	8	9	A	b	C	d	E	F
ディスプレイレジスタデータ	08	09	0A	0B	0C	0D	0E	0F
セグメントデータ	7F	6F	77	7C	39	5E	79	71
ディスプレイデータ	H	J	L	n	o	P	q	r
ディスプレイレジスタデータ	10	11	12	13	14	15	16	17
セグメントデータ	76	1E	38	54	5C	73	67	50
ディスプレイデータ	U	u	y	h	BLANK	-	.	
ディスプレイレジスタデータ	18	19	1A	1B	1C	1D	1E	
セグメントデータ	3F	1C	6E	74	00	40	80	

注）アドレスおよびデータディスプレイレジスタに1FH以上の値を入れてサブルーチンコールをすると、モニタの命令コードの一部をセグメントデータとみなしてデータ変換を行ってしまいます.

1000 0000
OR 0000 0000
1000 0000

(4)　フローチャート　図10.4にフローチャートを示します.

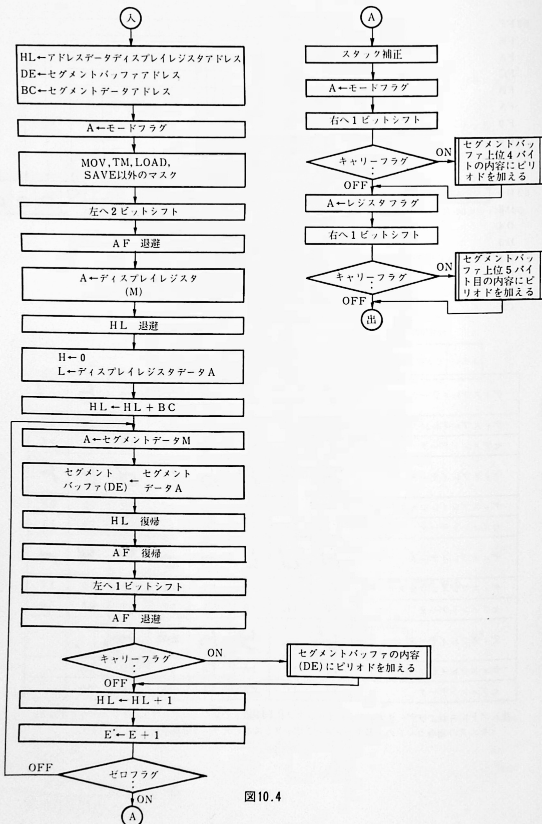


図10.4

(5) 使用例

```
          ORG   8000H
8000      MVI   B,8        06  08
          LXI   H,ADW1     21  D6  83
8005      MVI   M,1CH      36  1C
          DCX   H          2B
          DCR   B          05
          JNZ   8005H      C2  05  80
800C      XRA   A          AF
          PUSH  PSW        F5
800E      MVI   B,7        06  07
          LXI   H,ADW2     21  D5  83
          LXI   D,ADW1     11  D6  83
8016      MOV   A,M        7E
          STAX  D          12
          DCX   H          2B
          DCX   D          1B
          DCR   B          05
          JNZ   8016H      C2  16  80
          POP   PSW        F1
          STA   DDW4       32  CF  83
          PUSH  PSW        F5
          CALL  SEGCV      CD  6C  05
          LXI   H,60FFH    21  FF  60
8029      DCR   L          2D
          JNZ   8029H      C2  29  80
          DCR   H          25
          JNZ   8029H      C2  29  80
          POP   PSW        F1
          INR   A          3C
          PUSH  PSW        F5
          CPI   1FH        FE  1F
          JNZ   800EH      C2  0E  80
          JMP   800CH      C3  0C  80
ADW1      EQU   83D6H
ADW2      EQU   83D5H
DDW4      EQU   83CFH
SEGCV     EQU   056CH
          END
```

このプログラムは，モニタの用意したディスプレイデータをＬＥＤの右端から順番に表示させていくプログラムです．無限ループになっているので，まるでネオンサインのように，文字が流れていきます．

3．2．3　RGDSP

⑴　スタート番地　　01A1H 番地あるいは 07CFH 番地

⑵　入出力条件　　入力パラメータ　　モードフラグ（83DFH 番地），
レジスタフラグ（83DEH 番地）
出力パラメータ　　なし
使用レジスタ　　A，F，B，C，D，E，H，L

⑶　機　　能　　アドレスレジスタおよびデータレジスタにセットされているデータをディスプレイレジスタに転送し，さらに各データの上位4ビット，下位4ビットをおのおの16進数として各セグメントデータに変換して，セグメントデータバッファに転送します．

これにより，アドレスレジスタおよびデータレジスタにセットされているデータを，LEDディスプレイに表示させることができます．

各レジスタおよびLEDディスプレイにおけるデータ転送の関係は図10．7のようになっています．

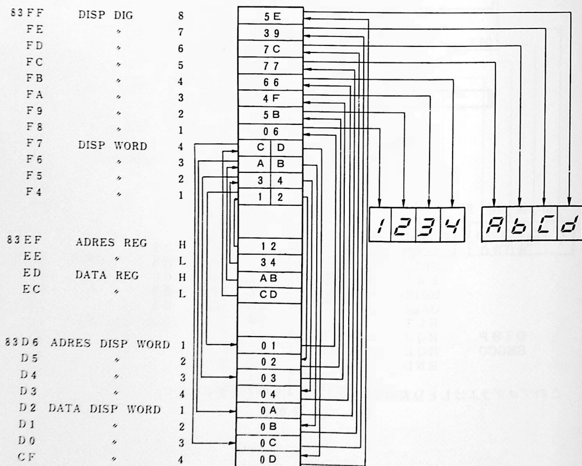


図10.7

(4) フローチャート　図10.8にフローチャートを示します.

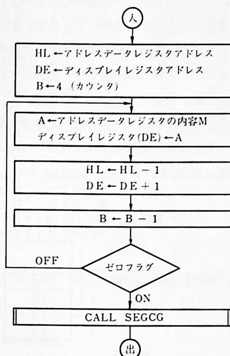


図10.8

(5) 使　用　例

```
        ORG     8000H
8000    LXI     H,1234H     21  34  12
        SHLD    ADRES       22  EE  83
        LXI     H,0ABCDH    21  CD  AB
        SHLD    DATA        22  EC  83
        CALL    RGDSP       CD  A1  01
        HLT                 76
ADRES   EQU     83EEH
DATA    EQU     83ECH
RGDSP   EQU     01A1H
        END
```

このプログラムはＬＥＤ表示部に 1 2 3 4 A B C D を表示します.

3.2.4 DLWCH

(1) スタート番地　　066ＤＨ番地

(2) 入出力条件　　入力パラメータ　　なし

出力パラメータ　　なし

使用レジスタ　　A, F, B, H, L

(3) 機　　能　　　　データレジスタの下位1バイトの上位4ビット，下位4ビットをおのおの16進数とみなして，ディスプレイレジスタデータに変換し，データディスプレイレジスタの下位2ワードに転送します．このサブルーチンは，LEDディスプレイの上位6桁（アドレスディスプレイおよびデータディスプレイの上位2桁）に表示するデータはそのままにしておき，LEDディスプレイの下位2桁（データディスプレイの下位2桁）に表示する16進データのみをデータディスプレイレジスタに転送する用途に使用します．

これは，LEDディスプレイの上位6桁に16進データと他の文字とを混在させて表示し，下位2桁に16進データを表示させようとする場合に便利です．

各レジスタのデータ転送の関係は図10．9のようになっています．

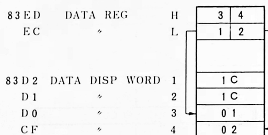


図10.9

(4) フローチャート　図10.10にフローチャートを示します．

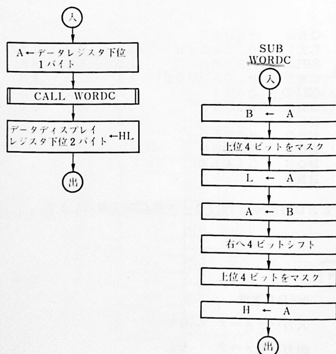


図10.10

(5) 使 用 例

```
             ORG    8000H
8000         MVI    A,85H      3E  85
             STA    DATA       32  EC  83
             LXI    H,0E17H    21  17  0E
             SHLD   ADW2       22  D5  83
             LXI    H,1714H    21  14  17
             SHLD   ADW4       22  D3  83
             LXI    H,171CH    21  1C  17
             SHLD   DDW2       22  D1  83
             CALL   DLWCH      CD  6D  06
             CALL   SEGCV      CD  6C  05
             HLT               76
ADW2         EQU    83D5H
ADW4         EQU    83D3H
DDW2         EQU    83D1H
DLWCH        EQU    066DH
SEGCV        EQU    056CH
             END
```

このプログラムはＬＥＤ表示部に Error 85 を表示します．

3.2.5 DWCH

(1) スタート番地　0664H番地

(2) 入出力条件　入力パラメータ　なし

出力パラメータ　なし

使用レジスタ　A，F，B，H，L

(3) 機　能　データレジスタ2バイトの各データ上位4ビット，下位4ビットをおのおの16進数とみなして，ディスプレイレジスタデータに変換し，データディスプレイレジスタ4バイトに転送します．

各レジスタのデータ転送の関係は図10.11のようになっています．

(4) フローチャート　図10.12にフローチャートを示します．

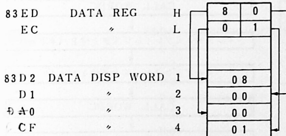


図10.11

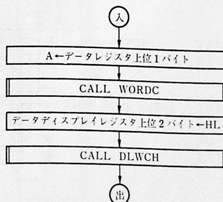


図10.12

(4) 使 用 例

```
                ORG   8000H
8000            LXI   H,8001H    21  01  80
                SHLD  DATA       22  EC  83
                LXI   H,1C15H    21  15  1C
                SHLD  ADW2       22  D5  83
                LXI   H,0C1DH    21  1D  0C
                SHLD  ADW4       22  D3  83
                CALL  DWCH       CD  64  06
                CALL  SEGCV      CD  6C  05
                HLT              76
DATA            EQU   83ECH
ADW2            EQU   83D5H
ADW4            EQU   83D3H
DWCH            EQU   0664H
SEGCV           EQU   056CH
                END
```

このプログラムはＬＥＤ表示部に［ ］［P］［C］［－］［8］［0］［0］［1］を表示します．

3.2.6 AWCH

(1) スタート番地　　0651H番地

(2) 入出力条件　　入力パラメータ　　なし

出力パラメータ　　なし

使用レジスタ　　A，F，B，H，L

(3) 機　　能　　アドレスレジスタ２バイトの各データ上位４ビット，下位４ビットをおのおの16進数とみなしてディスプレイレジスタデータに変換し，アドレスディスプレイレジスタ４バイトに転送します．

各レジスタのデータ転送の関係は図10.13のようになっています．

(4) フローチャート　図10.14にフローチャートを示します．

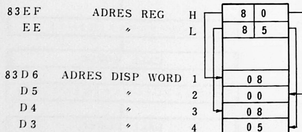


図10.13

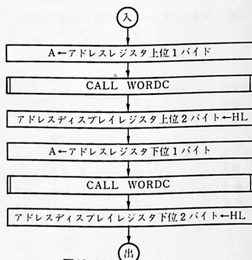


図10.14

(5) 使 用 例

```
                ORG    8800H
8000            LXI    H,8085H     21    85    80
                SHLD   ADRES       22    EE    83
                LXI    H,0C15H     21    15    0C
                SHLD   DDW2        22    D1    83
                LXI    H,181CH     21    1C    18
                SHLD   DDW4        22    CF    83
                CALL   AWCH        CD    51    06
                CALL   SEGCV       CD    6C    05
                HLT                76
ADRES           EQU    83EEH
DDW2            EQU    83D1H
DDW4            EQU    83CFH
AWCH            EQU    0651H
SEGCV           EQU    056CH
                END
```

このプログラムはＬＥＤ表示部に |8|0|8|5|C|P|U| | を表示します．

3.2.7 CH

(1) スタート番地　　064EH 番地

(2) 入出力条件　　入力パラメータ　　なし

出力パラメータ　　なし

使用レジスタ　　A，F，B，H，L

(3) 機　　能　　アドレスおよびデータレジスタ４バイトの各データ上位４ビット，下位４ビットをおのおの16進数とみなして，ディスプレイレジスタデータに変換し，アドレスおよびデータディスプレイレジスタに転送します．

各レジスタのデータ転送の関係は図10.15のようになっています．

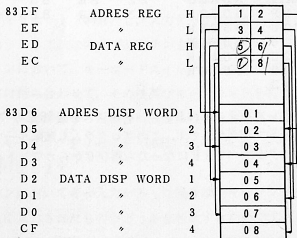


図10.15

(4) フローチャート 図10.16にフローチャートを示します.

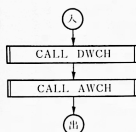


図10.16

(5) 使用例

```
        ORG   8000H
8000    LXI   H,0        21  00  00
        SHLD  ADRES      22  EE  83
        SHLD  DATA       22  EC  83
8009    CALL  CH         CD  4E  06
        CALL  SEGCV      CD  6C  05
        XRA   A          A7
8010    DCR   A          3D
        JNZ   $-1        C2  10  80
        LHLD  DATA       2A  EC  83
        INX   H          23
        SHLD  DATA       22  EC  83
        MOV   A,H        7C
        ANA   A          A7
        JNZ   8009H      C2  09  80
        MOV   A,L        7D
        ANA   A          A7
        JNZ   8009H      C2  09  80
        LHLD  ADRES      2A  EE  83
        INX   H          23
        SHLD  ADRES      22  EE  83
        JMP   8009H      C3  09  80
ADRES   EQU   83EEH
DATA    EQU   83ECH
CH      EQU   064EH
SEGCV   EQU   056CH
        END
```

このプログラムは，アドレスレジスタおよびデータレジスタを16進カウンタとして使い，それをLED上に表示させるプログラムです．このプログラムも無限ループになっており，アドレスおよびデータレジスタがすべてFFになったら再び0からカウントをやり直します．

4．入力用サブルーチン

4．1 機能説明

4．1．1 INPUT

(1) スタート番地　　0223H番地あるいは0690H番地

(2) 入出力条件　　入力パラメータ　○キーフラグ（83DDH 番地）

00H：直前のサブルーチンコールでキー入力無

FFH：直前のサブルーチンコールでキー入力無

出力パラメータ　○キー入力無

A＝FFH

キーフラグ＝00H

○キー入力有

A＝入力データ（16進データ）

キーフラグ＝FFH

使用レジスタ　A，F，B，D，E，H，L

(3) 機　　能　　キースイッチを8キーずつ三つのブロックに分け，各キーブロックを1回ずつスキャンして，キー入力の有無を調べています．具体的にはμPD8255（PPI）のPORT Aをスキャンデータの入力用に，またPORT Cの4，5，6の3ビットをキーブロックスキャン用に使用しています．各キーブロックをスキャンするときは，各キーブロックに対応したPORT CのビットをLowレベルにすることで，選択を実現しています．選択されたキーブロックの中に押されているキーボードスイッチがあった場合は，対応ビットのみLowレベルになって，そのキーブロックのデータがPORT Aに入力されます．

1回のキースキャン（3ブロックすべて）において，いずれのキーボードスイッチも押されない場合は，キーフラグをリセット（00H）し，入力データのパラメータとして用いているアキュムレータにFFHをセットして，このサブルーチンを抜け出します．

1回目のキースキャンにおいて，キーボードスイッチが押された場合は，チャタリングタイマ設定時間（約9ms）だけ待ち合わせて，その後再びキースキャンを行います．この2度目のキースキャンにおいて，キー入力が無かった場合（一度目のキースキャンにおける入力データはノイズ）は，先程と同様にキーフラグをリセット（00H）し，アキュムレータにFFHをセットして，このサブルーチンを抜け出します．

2回目のキースキャンにおいてキー入力があった場合は，キーフラグをセンスします．これは入力データが今回新たに入力されたものかどうかを判断するためです．したがってこのキーフラグセンスの結果，キーフラグがFFHである状態というのは，直前のサブルーチンコール（INPUT）における入力データと今回の入力データとがまったく同じものであることを意味しています．つ

まりずっとキーボードスイッチが押し続けられている状態を示しています．この場合はキースキャンを引き続き行い，キーボードスイッチが一度離されるまでこのサブルーチン内で待ち続けます．

キーフラグセンスの結果，キーボードスイッチが今回始めて押されたことが検出された場合は，アキュムレータに押されたキーボードスイッチに対応する16進データを格納し，またキーフラグをリセットした後，このサブルーチンを抜け出します．

このサブルーチンを利用すれば，プログラムに何らかの実行をさせながらキーの状態をモニタすることができます．

なおキーボードスイッチと変換される16進データとの関係は表10．1のようになっています．

表10．1

入力ポート＼スキャンポート	PC－4		PC－5		PC－6	
	入力キー	16進データ	入力キー	16進データ	入力キー	16進データ
PA－0	0／　AF	00	8／	08	RUN	10
PA－1	1／　BC	01	9／	09	CONT	11
PA－2	2／　DE	02	A／SAVE	0A	ADRES SET	12
PA－3	3／　HL	03	B／LOAD	0B	READ DEC	13
PA－4	4／　SP	04	C／T　M	0C	READ INC	14
PA－5	5／BR.P	05	D／MOV	0D	WR／ENT	15
PA－6	6／BR.D	06	E／OUT	0E	MODE	16
PA－7	7／	07	F／I　N	0F	REG	17

キースイッチに付けられている名前は，モニタプログラムで解釈される意味に合わせてありますが，キーの位置とその意味付けはプログラムの処理のみで決まるものですから，同じキーに違った意味を持たせて使用することもできます．

注）　チャタリング

メカニカルタイプのスイッチではその操作時に必ずチャタリングが生じます．チャタリ

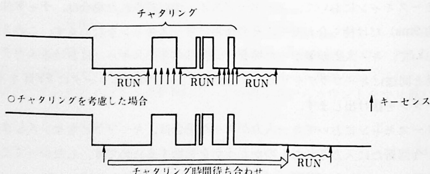


図10.17

ングを考慮しないでキースキャンを行うと一度しかキースイッチを押さないのに，２度以上押したように動作を行ってしまう可能性があります．このサブルーチンでは，チャタリングを考慮して，一度キー入力が検出されたら，チャタリングの時間を待ち合わせてから再びキーセンスを行っています（図10.17参照）．

(4)　フローチャート　図10.18にフローチャートを示します．

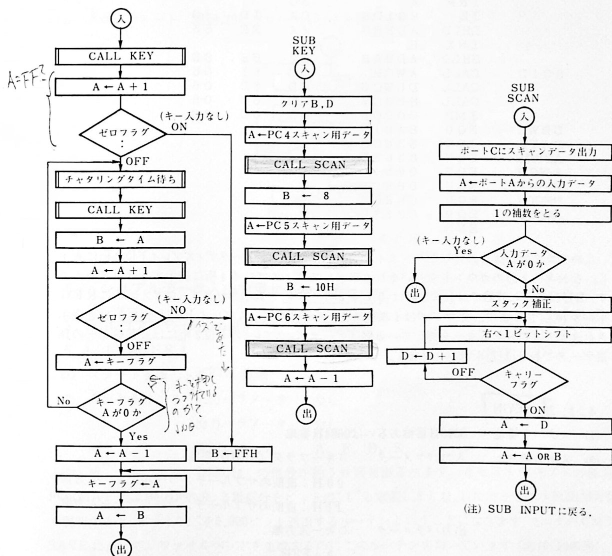


図10.18

(5) 使 用 例

```
        ORG   8000H
8000    LXI   H,101DH   21  1D  10
        SHLD  DDW2      22  D1  83
        LXI   H,0       21  00  00
        SHLD  ADRES     22  EE  83
800C    CALL  INPUT     CD  23  02
        STA   DATA      32  EC  83
        INR   A         3C
        JZ    801DH     CA  1D  30
        LHLD  ADRES     2A  EE  83
        INX   H         23
        SHLD  ADRES     22  EE  83
801D    CALL  AWCH      CD  51  06
        CALL  DLWCH     CD  6D  06
        CALL  SEGCV     CD  6C  05
        JMP   800CH     C3  0C  80
DDW2    EQU   83D1H
DATA    EQU   83ECH
ADRES   EQU   83EEH
AWCH    EQU   0651H
DLWCH   EQU   066DH
SEGCV   EQU   056CH
INPUT   EQU   0223H
        END
```

このプログラムは，入力したキースイッチの16進データをデータディスプレイ下位2桁に表示し，なおキー入力のカウントを行いそれをアドレスディスプレイ4桁に表示するプログラムです．RUNさせた状態ではLEDには[0][0][0][0][H][-][F][F]と表示されます．たとえば次にREGキーを押してみます．カウンタは1増加され，アドレスディスプレイ上には[0][0][0][1]と表示されます．またREGキーを押している間はデータディスプレイの下位2桁にはREGキーの16進データである[1][7]が表示されます．

4.1.2 KEYIN

(1) スタート番地　　0216H番地あるいは0684H番地

(2) 入出力条件　　入力パラメータ　　○キーフラグ（83DDH番地）

00H：直前のサブルーチンコールでキー入力無

FFH：直前のサブルーチンコールでキー入力有

出力パラメータ　　○キー入力無

A＝FFH

キーフラグ＝00H

○キー入力有

A＝入力データ（16進データ）

キーフラグ＝FFH

使用レジスタ　　A，F，B，D，E，H，L

(3) 機　　能　　このキー入力サブルーチンにおいては，キーボードスイッチが始めて押されたことが検出されるまで，このルーチンの中でキースキャンを繰り返します．サブルーチンINPUTと違うところは，サブルーチンINPUTはキースイッチが押されなくてもアキュムレータにデータFFHを格納してルーチンから抜け出てきたのに対し，このキー入力サブルーチンでは新たなキー入力が施されるまでは，キースキャンをサブルーチン内部で繰り返しているという点です．

(4) フローチャート　図10.19にフローチャートを示します．

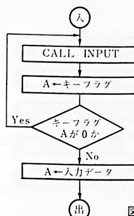


図10.19

5．シリアルデータ入出力用サブルーチン

5.1 シリアルデータ出力用サブルーチン機能説明

5.1.1 REDER

(1) スタート番地　　06E7H 番地

(2) 入出力条件　　入力パラメータ　　なし
出力パラメータ　　なし
使用レジスタ　　A，F，D，E

(3) 機　　能　　2,400Hz の信号が約5秒間送出されます．カセットテープへの録音FORMATで2,400Hzの波，2周期分をビットの“1”と定義しました．したがってこのFORMATで説明すると，ビット“1”を6,000ビット送出するルーチンということになります．モニタのCMT SAVEにおいて，ファイルヘッドを生成するのにこのルーチンを用いています（図10.20参照）．

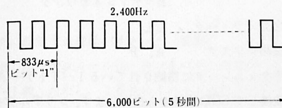


図10.20

(4) フローチャート　図10.21にフローチャートを示します.

(5) 使　用　例　　モニタサブルーチン SRIOT参照

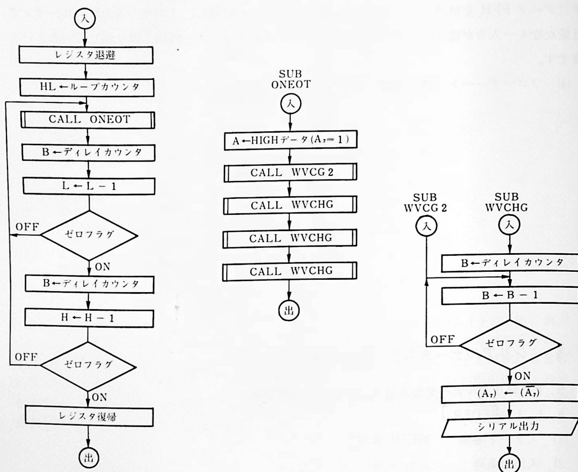


図10.21

5.1.2 SRIOT

(1) スタート番地　　0701H番地

(2) 入出力条件　　入力パラメータ　　A＝出力データ

B＝ディレイカウンタ

出力パラメータ　　なし

使用レジスタ　　A, F

(3) 機　　能　　アキュムレータに格納されている1バイトデータを, 下記FORMATに従ってシリアルデータに変換してSOD端子から出力します. シリアルデータは, 周波数2,400Hzの波2周期をデータ“1”とし, 周波数1,200Hzの波1周期をデータ“0”としています.

○データ転送速度　　1,200baud（bit/s）

○１ビットデータ形式

"０"…1,200Hz　１サイクル

"１"…2,400Hz　２サイクル



○１バイトデータ形式（１バイトデータ73Hの場合）：SOD 出力波形

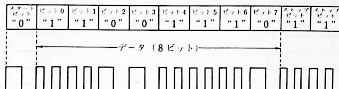


図10.22

ハードウェアの簡略化を図るため，カセットテープへの出力波形はすべてソフトウェアで生成しています．したがってユーザーがこのルーチンを使用する場合にはいくつかの法則に従わなければなりません．

１バイトデータを出力する場合は，このルーチンを呼ぶことによって波形生成は自動的に行われます．しかし，READER 出力ルーチンを呼んだ後１バイト出力ルーチンを呼ぶとか，１バイト出力ルーチンを呼んだ後再び１バイト出力ルーチンを呼ぶという，いわゆる接続部分はユーザーが管理しなければなりません．この接続部分のしわ寄せはすべて２番目に呼ばれる１バイト出力ルーチンのスタートビットが請負います．この接続部分，つまりスタートビットの波形を正常な大きさで生成するためには１バイト出力ルーチンを呼ぶ前に正常な遅延時間を作る適当なパラメータをＢレジスタに設定しなければなりません．

では適切なパラメータの設定方法を二つの場合に分けて説明しましょう．

① READER 出力ルーチンをコールした後，１バイト出力ルーチンをコールする場合の設定方法

n：Ｂレジスタに設定する遅延パラメータ（整数）

X：READER 出力ルーチン後１バイト出力ルーチンをコールするまでに要したステート数．ただし，１バイト出力ルーチンコール命令も含む．

$$n=\frac{1}{14}(809-X)$$

② １バイト出力ルーチンをコールした後，再び１バイト出力ルーチンをコールする場合の設定方法

n：Ｂレジスタに設定する遅延パラメータ（整数）

X：１バイト出力ルーチンコール後，再び１バイト出力ルーチンをコールするまでに要したステート数．ただし，２番目の１バイト出力ルーチンコール命令も含む．

$$n=\frac{1}{14}(852-X)$$

(4) フローチャート　図10.23にフローチャートを示します.

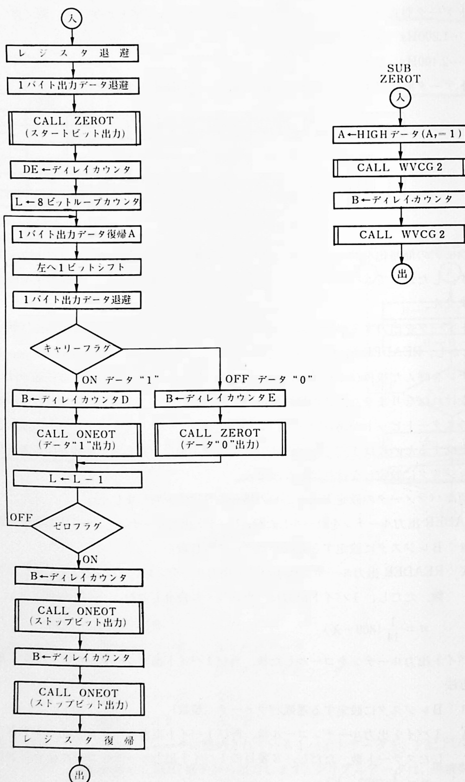


図10.23

(5) 使 用 例

```
                ORG    8000H          ステート数
                LXI    H,8000H
                MVI    C,0
                CALL   REDER
           *1   CMP    M               (7)  ┐
                MVI    B,PARA1          7   │ ┐
  LOOP:         MOV    A,M              7   │ │ X1
                CALL   SRIOT           18   ┘ │
           *2   SHLD   SRIOT           (16)   │
                MVI    B,PARA2          7     │ X2
                INX    H                6     │
                INR    C                4     │
                JNZ    LOOP            10     ┘
                HLT
  PARA1         EQU    55
  PARA2         EQU    56
  REDER         EQU    06E7H
  SRIOT         EQU    0701H
                END
```

このプログラムは8000H番地から 80FFH 番地までの内容をREADER送出後，シリアル転送を行うプログラムです．ここではパラメータの設定方法およびダミー命令の挿入方法を学んで下さい．

$$X_1 = 7 + 7 + 18 + (7)^{*1} = 39$$

$$\therefore \text{PARA } 1 = \frac{1}{14}(809 - X_1) = 55$$

$$X_2 = 7 + 6 + 4 + 10 + 7 + 18 + (16)^{*2} = 68$$

$$\therefore \text{PARA } 2 = \frac{1}{14}(852 - X_2) = 56$$

PARA 1，および PARA 2 は整数でなければなりません．したがって＊1および＊2は，ステート数補正のために挿入されたダミー命令です．ダミー命令を挿入する場合は，プログラム本体に影響を及ぼさない命令でなければなりません．

5.2 シリアルデータ入力用サブルーチン機能説明

5.2.1 RDSCH

(1) スタート番地　0765H番地

(2) 入出力条件　入力パラメータ　なし

出力パラメータ　なし

使用レジスタ　A，F

(3) 機　能　いったんこのサブルーチンに入ると，2,400Hz の信号が256周期連続して検出されるまで，このサブルーチンを抜け出しません．モニタではカセットテープに作成したファイルヘッド部の検出にこのサブルーチンを使用しています．

(4) フローチャート　図10.24にフローチャートを示します．

(5) 使 用 例　モニタサブルーチンSRIIN参照．

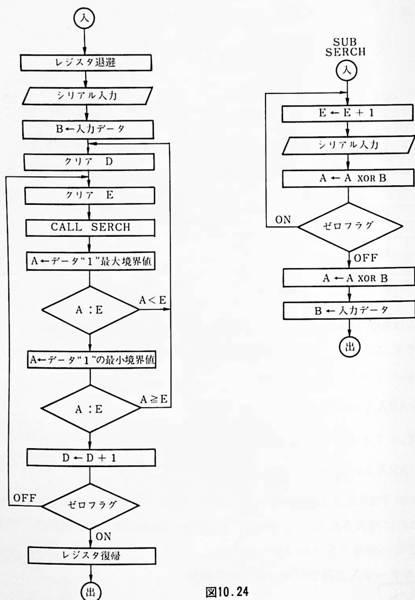


図10.24

5.2.2　SRIIN

(1)　スタート番地　　0783H番地

(2)　入出力条件　　入力パラメータ　　なし

出力パラメータ　　A＝入力データ

使用レジスタ　　A，F

(3)　機　　能　　SID端子より入力されるシリアルデータを受信して8ビットデータに編集して，アキュムレータに格納します．

1バイトデータのFORMATは1バイトデータ出力サブルーチンで述べた通りです．したがってこのサブルーチンに入るとまずスタートビットの検出を行います．

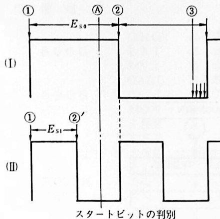


図10.25

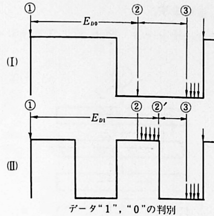


図10.26

図10.25において(Ⅰ)はスタートビットの場合，(II)はREADER部の場合です．まず①の変化点が検出されると，カウンタEがクリアされ再び変化点が来るまでカウントアップを始めます．(Ⅰ)の場合は②の変化点までカウントアップされます．②までカウントアップされたカウンタをE_{S0}とします．(II)の場合は②′の変化点までカウントアップされます．この場合のカウンタの値をE_{S1}とします．図から明らかなように，E_{S0}とE_{S1}との関係は$E_{S0}>E_{S1}$となっています．したがってこれらの値の間に両者を見分ける境界値を設けて，この境界値（Ⓐ）とカウンタとを比較すれば，スタートビットかどうかの判別はつくわけです．

(Ⅰ)の場合のように，スタートビットが検出されると，その後②から③の位置まで遅延動作を行ない再び変化点検出のスキャンを始めます．この遅延動作はスキャンを行う時期をできるだけ次の変化点に近づけ，ノイズによる誤判定を防ぐためのものです．

スタートビット判別後は，シリアルデータの読み込みを行い，１バイトデータに編集しなければなりません．ここでは編集以前の問題であるデータの判別方法を述べます．

図10.26においてスタートビット判別方法と同様に①の変化点においてカウンタEはクリアされます．その後，②の時点まではカウンタEのカウントアップだけを行い変化点検出は行いません．②の時点まできて始めて，変化点検出を始めます．データが"0"の場合はこの時点でデータが変化しているわけですから，ここの値がそのままカウンタの値E_{D0}となります．ところがデータが"1"の場合はその後もカウントアップを続け，②′にきて始めてカウントアップを終了します．したがってカウンタの値をみれば②の時点でカウントを終了したものか，あるいはその後もカウントアップを続けたものであるかは一目瞭然です．

データ判別後再び次のデータの判別に移るわけですが，この場合も③の時点まで遅延動作を行い，ノイズによる誤判定を防いでいます．

このSRIINルーチンを用いるときに注意することは，SRIINルーチンを複数回コールする場合，それらの間が4,000ステート以上にならないことです．4,000ステート以上の時間が経過すると，次の１バイトデータのスタートビットが始まってしまい，データの読み込みが間に合わなく

なってしまいます.

(4) フローチャート　図10.27にフローチャートを示します.

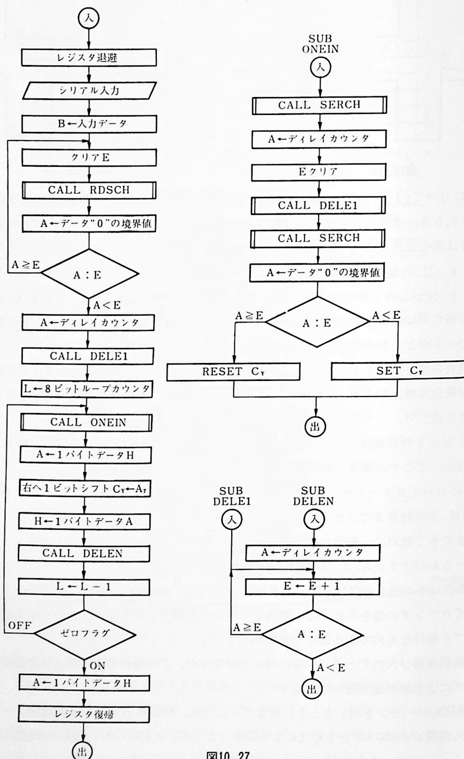


図10.27

(5) 使用例

```
                ORG    8000H
8000            LXI    H,8100H    21   00   81
                MVI    B,0        06   00
                CALL   RDSCH      CD   65   07
8008            CALL   SRIIN      CD   83   07
                MOV    M,A        77
                INX    H          23
                INR    B          04
                JNZ    8008H      C2   08   80
                HLT               76
RDSCH           EQU    0765H
SRIIN           EQU    0783H
                END
```

ＲＥＡＤＥＲ以降次々と転送されてくるデータを，8100H番地から81ＦＦH番地までに格納します．

11章　TK-85のハードウェア

1．はじめに

本章には，TK-85の回路を機能別に分割し，その個々の機能ブロックについてシステム内での役割りおよびその動作についての説明が述べられています．

2．TK-85 システム構成

TK-85 は，CPU システムとして必要最小機能により構成されており，また今後 TK-85 を機能拡張するための必要なデータバス，アドレスバスおよびコントロール信号がカードエッジに出力されています．

まず，TK-85 のシステムブロック図によりシステム全体の概要を頭に入れ，これから個々の

図11.1　システムブロック図

項目で述べられる回路のシステムにおける位置付けを理解して下さい．図11．1にTK-85のシステムブロック図を示します．

TK-85は大きく分けて，CPU部，メモリ部，キー入力部，表示部，CMTインターフェイス部の五つのシステムブロックにより構成されています．CPU部は，μPD8085A CPUを中心として構成されており，システム全体を制御する心臓部であるといえます．

またTK-85のCPUは，約2.5MHzのクロックで動作しています．メモリ部には，ICメモリが使用されており，ROM（Read Only Memory）およびRAM（Random Access Memory）が搭載されています．ROMにはモニタプログラムが格納されており，電源投入時およびRESETキーあるいはMONキーが押されたときに走りはじめます．RAMは，1kバイト搭載されており，一部をモニタプログラムがワーキングエリアとして使用しているほか，プログラムやデータエリアとして使用することができます．

キー入力部は，その制御用にプログラマブル周辺インターフェイスLSI（μPD8255AC-5)が使われており，これによりTK-85に対するプログラミングおよびその実行，デバッグ等を行う24個のキーボードスイッチがスキャンされます．

表示部は8個の7セグメントLED（発光ダイオード）ディスプレイで構成されており，16進数によりアドレスやデータを表示することができます．表示される内容はプログラムによる転送ではなく，サイクルスチールによるDMA転送によって，常時自動的に転送・表示が行われています．

CMTインターフェイス回路はμPD8085Aのシリアル入出力端子を利用してデータの入出力を行っています.この回路により市販のオーディオテープレコーダを外部記憶装置として用いることができ，自分の作成したプログラムおよびデータをセーブおよびロードすることができます．

下記にTK-85の仕様を示します．

〈TK-85 仕様〉

CPU　：μPD8085A
クロック周波数：2.4576MHz（4.9152MHzクリスタル使用)
ROM　：μPD2316EC（μPD2716Dピンコンパチブル)
　※実装2kバイト（MAX．8kバイト)
RAM　：μPD2114LC　1kバイト
入力装置：キーボードスイッチ　25個
表示装置：8桁7セグメント LEDによる16進数表示
パラレル I/Oポート：μPD8255AC-5の8ビットおよび4ビットポートをカードエッジに出力（入力，出力プログラム可能)
バ　ス：TK-80　バス（上位コンパチブル)

CMT 転送レート：1200ボー
電　源：外部電源が必要　＋5V±5％　単一電源
消費電流：1.2A以下
寸　法：310×230mm

3．アドレスバス，データバス

TK-85 で使用されていますμPD8085A はマルチプレクスの方式のデータバスを採用しています．

μPD8085Aのアドレス信号は，上位8ビットが直接CPUの端子より出力されており，下位8ビットはデータバス上にCPU がデータの入出力を行っていないタイミングに出力されます．

データバス上にCPU が下位アドレスを出力している期間中，CPU のコントロール信号 ALE は High となり下位アドレスを出力中であることを示します．この下位アドレスはμPB8212 にこのタイミングでラッチされます．

また上位アドレスは 74LS367（トライステートバッファ）でドライブされています．TK-85 のアドレスバスはこの 74LS367 より出力される上位アドレスとμPB8212より出力される下位アドレスにより形成されています．このうちA0，A1，A2，は7407（オープンコレクタバッファ）を介して出力されています．この理由は7節で述べます．

7節で述べる DMA 転送（Direct Memory Access）が行われる期間，アドレスバスはフローティング状態になります．ここでA0，A1，A2のバッファ 7407 は入力がハイ インピーダンスによる出力不安定となるのを防ぐため入力側ラインは 1kΩ の抵抗 によりプルアップされています．

TK-85のアドレスバスは，HALT時およびHOLD時にフローティング状態にはなりませんので（DMAが行われているため），外部装置においてDMA 転送等を行う場合は，アドレスバスに外部装置の入力側でトライステートバッファを入れる必要があります．TK-85のデータバスは，μPB 8216C（オプション）によりドライブされカードエッジに出力されています．このため，外部拡張を行うためには，IC 24およびIC 25のIC ソケットに μPB8216C を挿入する必要があります．

データバスは，TK-85内のROM，RAMおよびI/O が選択された場合およびRESET時，HALT時およびHOLD時に外部とは遮断状態になります．

〈ボード上に μPB8216を2個挿入する場合，IC の向きに注意して下さい．〉

4．ROM，RAM の構成

CPU システムにおいて ROM，RAM はプログラムやデータの記憶および CPU レジスタ内容の一時退避などに用いられ重要な構成要素となっています．

図11．2 は TK-85 の ROM，RAM 回路部分を示しています．ROMはボード上最大 8 k バイト実装が可能で，μPD2716 が 4 個使用できます．TK-85 ではすでに 0 H～7FFH 番地のエリアにモニタプログラムが常駐するようにマスク ROM μPD2316EC（μPD2716 ピンコンパチブル）が実装ずみです．

RAM はμPD2114LC（スタティック）が 2 個実装されていて，1 k バイトの容量です．

ROM，RAM のそれぞれの IC はチップセレクト端子（μPD2316 では $\overline{CS3}$，μPD2716 では $\overline{OE}$ 端子）が Low レベルになることにより選択されます．このチップセレクト信号は CPU より出力される IO/$\overline{M}$，$\overline{RD}$，$\overline{WR}$ と，ROM，RAM に入力されない残りのアドレス信号，それぞれ 5 本・6 本とを組み合せて作られます．74LS139 によって出力されるアドレスデコードされたチップ

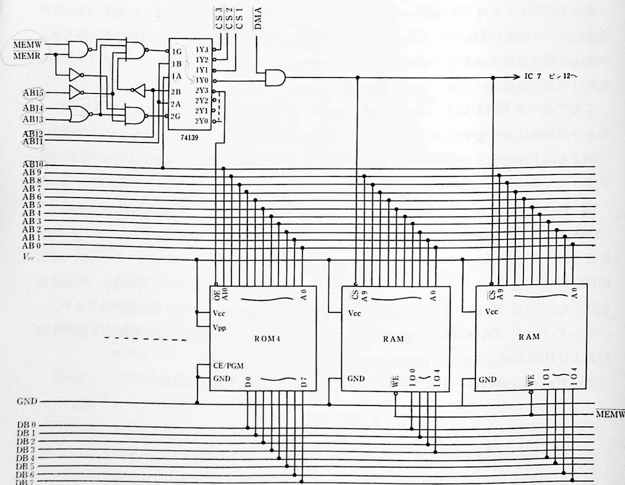


図11．2　ROM，RAM 回路

セレクト信号のうち，8400H～87FFH，8800H～8BFFH，8C00～8FFFHを示す．チップセレクト信号（$\overline{CS1}$，$\overline{CS2}$，$\overline{CS3}$）は基板のカードエッジを通して外部へ出力されます．この信号を用いてRAM増設をすることができます．（12節参照）

また別に外部でROM，RAMおよびI/Oを増設する場合，外部で使用するコントロール用信号もカードエッジに設けてあります．

5．リセット回路

TK-85ではパワーオンと同時にCPUに対してRESET信号が発生し，CPUはRESET信号解除後に0番地より書き込まれているモニタプログラムを実行しはじめます．この動作を行うのが図11．3で示すリセット回路です．また図11．5で示すタイミングチャートを見て下さい．

図11．5のAで示される電圧変化によって電源が投入された場合，CPUの$\overline{\text{RESET IN}}$端子に加えられる信号変化をBで示してあります．このBの信号変化はリセット回路のR_2とC_1とが持つ時定数によりAで示される電源電圧が積分されたものです．CPUの$\overline{\text{RESET IN}}$の内部にはシュミット回路が組み込まれており，入力されるBの信号波形はCPU内ではCで示される信号として検出されます．つまり，図11．5のタイミングチャートにおいて電源投入時TよりT′までの

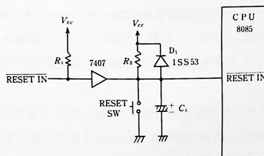


図11．3　リセット回路

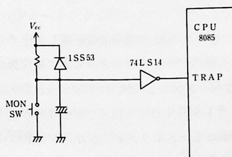


図11．4　トラップ回路

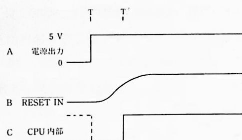


図11.5 電源 ON 時リセット回路タイミングチャート

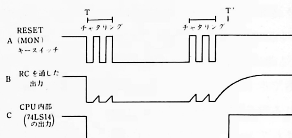


図11.6 RESET (MON) スイッチ ON 時回路タイミングチャート

期間CPUはリセット状態でいることになります．これによりCPUはリセット解除後，モニタプログラムを確実に実行しはじめます(この動作をパワーオンリセットと呼びます)．

また，プログラム実行中任意にリセットを行うためにリセット回路にはRESETスイッチが付いています．RESETスイッチを押すことによりCPUの $\overline{\text{RESET IN}}$ 端子に入力される信号は強制的にLowレベルになります．

RESETスイッチによりリセット信号を発生させた場合，スイッチより出力される信号は図11.6で示されるタイミングチャートのAと同様なものとなります．Aにおいてスイッチを押した直後，およびスイッチを離した直後にチャタリングという現象が発生します．チャタリングを含むリセット信号が直接CPUに入力された場合，RESETスイッチを1回押すごとにリセット信号が複数回CPUに入力されることになってしまいます．この現象を防ぐためにも抵抗 R_2 とコンデンサ C_1 が働いています．

図11.6のBはスイッチの出力がCRの積分回路を通して出てくる信号波形です．RESETスイッチを押した直後と離した直後のチャタリングはCRの時定数によりスレッシュホールドレベルまで達せずCPU内部では信号変化として検出されません．このようなCRの働きによりRESETスイッチを1回押すことにより1発のリセット信号を発生させることができます．

また，このリセット回路にはカードエッジより入力される $\overline{\text{RESET IN}}$ 信号によっても，CPUに対するリセット信号を発生させられるようになっています．

6．トラップ回路

TK-85のトラップ回路はMONキースイッチが押された場合，CPUがどのようなプログラムを処理していてもモニタプログラムに戻るようにするための回路です．図11．4はTK-85で使用されているトラップ回路です．リセット回路と同様の構成となっていますが，ただ一つ違う点はシュミット回路として74LS14がCPUに外付けとなっていることです．

μPD8085AのTRAP端子は割り込みをCPUに対して要求するための端子の一つで，ノンマスカブルインターラプト（ソフトウェアにより禁止できない割り込み）となっています．CPUがTRAP要求を受け付けると，プログラム実行はリスタートアドレス24Hに飛び，次にTRAP処理プログラムを実行した後，モニタに戻ります．

7．表示回路とDMA転送

TK-85において，人間と情報をやり取りする窓口としてデータ・命令を入力する役目を果しているのがキーボードスイッチであり，CPUよりのデータ表示出力をする役目を果しているのが7セグメントLED 8個です．このLEDディスプレイを含むデータの各制御および動作を行うための回路が表示回路です．ここで，RAMより直接表示データを表示回路へ送る方法としてDMA転送が行われています．

TK-85のDMA転送は，サイクルスチールによって行われており，このためにCPUを一時停止状態（HOLD状態）にすることは行っておりません．

CPU μPD8085が命令を実行してゆく過程において，必ずある一定期間データバスもアドレスバスもCPUが使用しないタイミングが存在します．

これはCPUタイミングのM1（マシンサイクル1）のT4ステートに当ります．

TK-85では，このタイミングをハードウェアにより検出して，このときにメモリと表示回路との中でDMA転送しています．

図11．7はTK-85で使用されている表示回路の部分を示しています．表示回路においてDMA転送時にシステムを制御するための基本となる信号がゲートG1より出力される$\overline{\text{DMAC}}$です．この信号はプログラム実行中，OPコードフェッチサイクル*時ごとに出力されています．

図11．8にOPコードフェッチサイクル時のDMA転送タイミングチャートが示してあります．この図において$\overline{\text{DMAC}}$信号は$\overline{\text{RD}}$の信号が立ち上がった後Lowになり，次のALE信号およびM1サイクル信号が出力されると同時にHighに戻ります．$\overline{\text{DMAC}}$信号がLowである期間CPUはOPコードを，解析，内部実行を行っており，CPUは外部の回路に対してアクセスしていない期間となります．つまりCPUが外部状況を見ていない期間に表示回路はアドレスバ

＊μCOM-85ユーザーズマニュアル（IEM-618B）参照

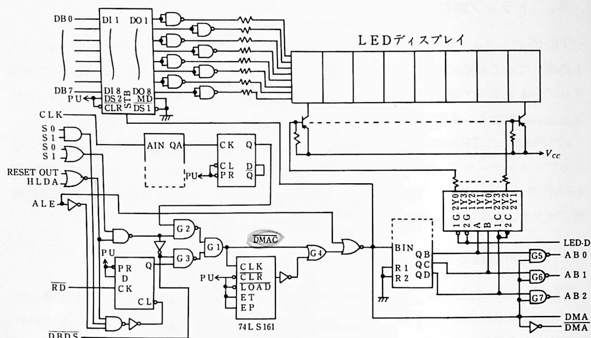


図11.7　表示回路

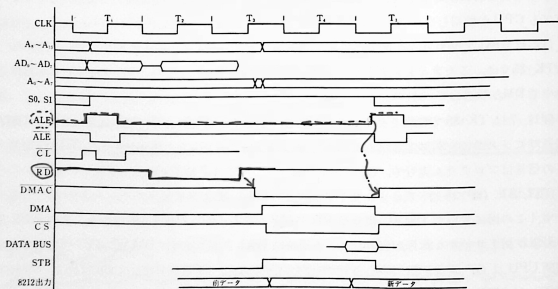


図11.8　DMA 転送タイミングチャート

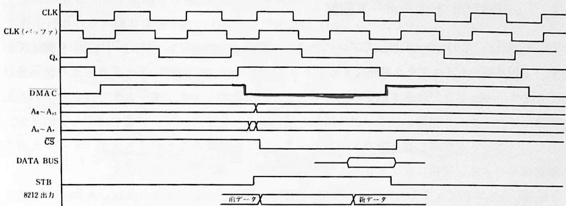


図11．9　DMA 転送（RESET，HOLD）

ス，データバスを利用して表示データを RAM より得ます．この動作をサイクルスチールというわけです．

プログラム実行中の $\overline{\text{DMAC}}$ 信号はゲートＧ３およびゲートＧ１を通って出力されますが，CPU が RESET，HOLD，HALT 状態の場合，$\overline{\text{DMAC}}$ 信号はゲートＧ２およびゲートＧ１を通って出力されます．この $\overline{\text{DMAC}}$ 信号は CPU クロック（CLK 信号）を４分周したものが使用されています．図11．9にこのときのタイミングチャートを示します．

ゲートＧ１より出力される $\overline{\text{DMAC}}$ 信号は，次に74LS161により16分周され出てくる16回に１回のCarry信号とともにゲートＧ４に入力します．Ｇ４より出てくる DMA 信号とそれを反転させた $\overline{\text{DMA}}$ 信号が実際に各回路の制御信号として利用されています．DMA 信号はカウンタに入力され，このカウンタにより DMA 転送時の RAM アドレスの下位３ビットデータを作り出しています．このアドレス下位３ビットデータは DMA 転送開始と同時にゲートＧ５，Ｇ６，Ｇ７より出力されます．このとき OP コードフェッチ時のアドレスデータが μPB8212 よりまだ出力されている可能性がありこの場合，DMA 転送用アドレスデータとぶつかることになってしまいます．このため，μPB8212より出力されているアドレスの下位３ビットは３節で述べられているようにオープンコレクタバッファ7407でドライブされており，データのぶつかり合いが起ってもシステムを破壊しないようになっています．

DMA 転送用アドレスは，アドレスバスの AB0H～AB9H までプルアップされており，DMA および $\overline{\text{DMA}}$ の各信号により制御され，ハイインピーダンス状態になったときに下位３ビット以外は High レベルとなります．また DMA 転送時に TK-85 ボード上の実装 RAM が選択されますので，プルアップと組み合わさってアドレスが 83F8H～83FFH まで選択されることになります．

8．μPD8255 とキーボード回路

TK-85 では，I/O ポートとしてプログラマブル周辺インターフェイス μPD8255 を備えています．μPD8255 はプログラム可能な 8 ビット I/O ポートを三つもち，モード 0 ～ 2 の使い分けができます．モード 0 では基本的な入出力ポートとして動作します．モード 1 ではデータの入出力制御にコントロール信号，ステータス信号を用いるモードです．モード 2 ではデータの入力，出力に同一ポートを使用し双方向の入出力ポートとして使用することができます．データ入出力の制御はモード 1 と同様にコントロール信号およびステータス信号を用います．

TK-85 のモニタプログラムでは，μPD8255 をモード 0 としてプログラムし，キーボードスイッチのデータ入力用としてポート A とポート C の一部を使用しています．またポート C の PC7 は 9 節で述べます 1 インストラクションステップ動作回路の制御に使われ，残りのポートはカードエッジに接続されています．

μPD8255 はモニタプログラムによりポート A のビットを入力状態に，またポート C の上位 4 ビットを出力状態にしてあります．このポート A の 8 ビットとポート C の上位 3 ビットのラインはマトリクス構造となっていて，その各交点に図11.10で示すようにキースイッチが結線されています．

ポート A の 8 ビットラインは抵抗 33 kΩ によりプルアップされています．キースキャン方法として TK-85 ではポート C の PC4，PC5，PC6 の各ビットに順次 Low レベルを出力します．ポ

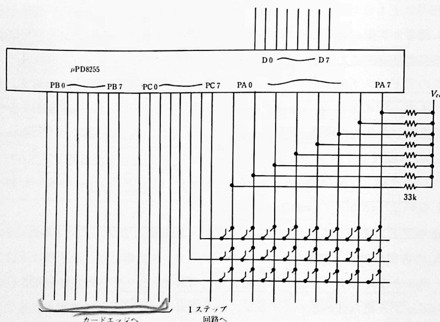


図11.10　キーボード回路

ートCの各ビットを変化させるごとにポートAのデータをCPUに読み込みます．キースイッチがどれも押されない限りポートAの8ビットラインは High レベルであり，入力データは FFH となります．またキースイッチが押された場合，押されたキースイッチに当るポートAのビットが Low レベルとなるため FFH とはなりません．つまり Low レベルにしたポートCのビットと，そのとき読みとったポートAのデータをみてやることにより，どのキースイッチが押されたかが判断できます．

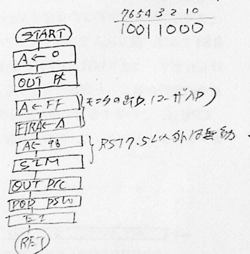
9．1インストラクションステップ動作回路

TK-85では自作プログラムの解析，検討およびプログラムミスの発見を容易にするために1インストラクションステップ動作回路が備えてあります．

TK-85の1インストラクションステップ動作回路では割り込みを利用して動作を行い，モニタプログラムにより途中レジスタおよびメモリの内容を読み出し，変更できるようになっています．この項目ではハードウェアの動作について述べます．

まずモニタプログラムより RUN キーおよび CONT キーが押されユーザープログラムにジャンプする部分のプログラムを下記に示します．

```
 1.  CONT:  XRA   A
 2.         OUT   PORTC
 3.         DCR   A
 4.         STA   FTRAP
 5.         MVI   A, MSKRS
 6.         SIM
 7.         OUT   PORTC
 8.         POP   PSW
 9.         EI
10.         RET
```



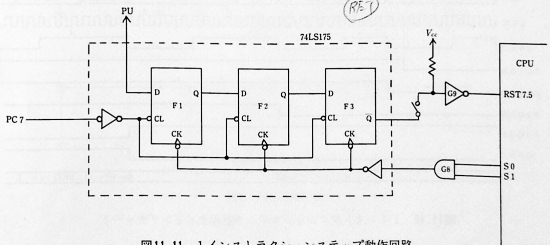


図11.11　1インストラクションステップ動作回路

ここで CPU が

1．　XRA

2．　OUT　　PORTC

の2命令を実行するとポートCの最上位ビット，7ビット目がLowとなり図11.11において各フリップフロップにクリアがかかります．ここでスライドスイッチを STEP 側に倒してある場合，TRAP 端子には Low が入力されます．

以下，次のポート動作時までクリアは解除されません．下記で示す命令以下の CPU 動作を図11.11とタイミングチャート図11.12を用いて述べます．

7．　OUT　　PORTC

この命令を実行することによりフリップフロップのクリアは解除され，ゲートG8より出力されるOPコードフェッチサイクルをカウントしはじめます．図11.12の1ステップ動作タイミングチャートを見ると，クリア解除後各フリップフロップはOPコードフェッチ信号を入力するたびに順番に出力が変化してゆき最後に時間Tにおいてフリップフロップ F3の $\overline{Q}$ が High からLowに変化します．

フリップフロップ F3のQはインバータを介してCPU のRST 7.5入力に接続されています．RST 7.5は，割り込み要求端子でこの入力信号の立ち上がりのエッジで，割り込み要求が受け付けられます．つまり図11.12でわかるように，割り込みはRET 命令の後ジャンプした先の初めの命令の実行が始まった直後に発生することになります．

CPUは，この第1番目の命令実行後割り込み処理モニタにジャンプしてきます．またCONTキーを押すことによって，同様な動作を再び行い割り込み処理モニタにジャンプしてきます．

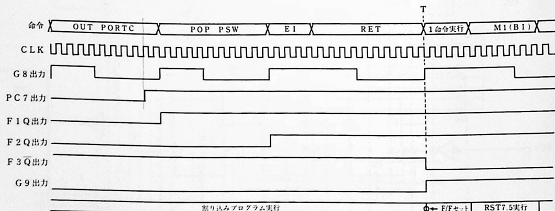


図11.12　1インストラクションステップ動作タイミングチャート

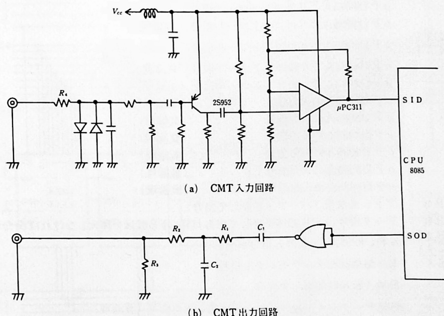
10. CMT 回路

TK-85 ではプログラムやデータを記憶する装置として市販オーディオテープレコーダを利用できます．このテープレコーダと CPU とのインターフェイスとして CMT 回路があります．

CMT 入力および出力回路はそれぞれ出力される信号を相手側に入力しやすいレベルおよび波形にします．

図11.13の(b)に CMT 出力回路を示します．μPD8085A のシリアル出力端子 SOD より出力される信号はC_2とR_1により積分されます．次にR_2とR_3によりテープレコーダのマイクロホン端子に入力できるレベルまで分圧しミニプラグに出力します．

図11.13の(a)は CMT 入力回路です．この回路では，市販オーディオテープレコーダより入力されるアナログ波形を整形しディジタル波形にしなければなりません．このため入力信号は保護抵抗R_4を通った後，ダイオード2個によりダイオードの順方向電圧0.8Vまでスライスされます．スライスされた信号は次段のアンプにより増幅されます．増幅された信号はコンパレータ(μPC311)により基準電圧と比較され，比較結果として出力に＋5Vと0Vのディジタル信号が出力されます．このディジタル信号は CPU の SID 端子に入力され，正確に0と1の判断を CPU が行います．



(a) CMT入力回路

(b) CMT出力回路

図11.13 CMT回路

11．カードエッジ信号表

ピン	A	B	ピン	A	B	ピン	A	B
1	GND	GND	18			35	PB 5	RST6.5
2	GND	GND	19			36	PB 6	NC
3	+5V	+5V	20		$\overline{\text{MEMR}}$	37	PB 7	NC
4		NC	21		$\overline{\text{MEMW}}$	38	PC 0	NC
5		NC	22	READY		39	PC 1	NC
6	$\overline{\text{ALE}}$	NC	23			40	PC 2	NC
7		NC	24			41	PC 3	NC
8	$\overline{\text{RD}}$	NC	25	$\overline{\text{HOLD}}$	$\overline{\text{HLDA}}$	42	$\overline{\text{CS 3}}$	$\overline{\text{INTR}}$
9	$\overline{\text{WR}}$	IO/$\overline{\text{M}}$	26		DB 7	43	$\overline{\text{CS 2}}$	INTR
10	AB15	AB 7	27	DMA	DB 6	44	$\overline{\text{CS 1}}$	RESET OUT
11	AB14	AB 6	28	$\overline{\text{DMA}}$	DB 5	45	$\overline{\text{RESET IN}}$	$\overline{\text{RESET OUT}}$
12	AB13	AB 5	29	$\overline{\text{DBSL}}$	DB 4	46		
13	AB12	AB 4	30	PB 0	DB 3	47		
14	AB11	AB 3	31	PB 1	DB 2	48	CLK	S0・S1
15	AB10	AB 2	32	PB 2	DB 1	49		S0+S1
16	AB 9	AB 1	33	PB 3	DB 0	50	GND	GND
17	AB 8	AB 0	34	PB 4	$\overline{\text{RST 5.5}}$			

〈各信号名の説明〉

$\overline{\text{ALE}}$	μPD8085Aよりのコントロール信号（負論理）
$\overline{\text{RD}}$	μPD8085Aよりのコントロール信号（負論理）
$\overline{\text{WR}}$	μPD8085Aよりのコントロール信号（負論理）
IO/$\overline{\text{M}}$	μPD8085Aよりのコントロール信号
$\overline{\text{HLDA}}$	μPD8085Aよりのコントロール信号（負論理）
RESET OUT / $\overline{\text{RESET OUT}}$	μPD8085Aよりのコントロール信号
$\overline{\text{INTA}}$	μPD8085Aよりのコントロール信号（負論理）
CLK	μPD8085Aよりのコントロール信号（正論理）
READY	μPD8085Aの入力信号（正論理）
$\overline{\text{RESET IN}}$	μPD8085Aの入力信号（負論理）
$\overline{\text{HOLD}}$	μPD8085Aの入力信号（負論理）
$\overline{\text{RST5.5}}$	μPD8085Aの入力信号（負論理）
$\overline{\text{RST6.5}}$	μPD8085Aの入力信号（負論理）
$\overline{\text{INTR}}$	μPD8085Aの入力信号（負論理）
AB15～AB0	アドレス信号（バッファを介して出力）
DB7～DB0	データ信号（μPB8216を介して出力）〔注 μPB8216を挿入しなければ出力されません〕
PB7～PB0 / PC3～PC1	μPD8255AC-5の入出力ポート
$\overline{\text{CS3}}$～$\overline{\text{CS1}}$	RAM増設用チップセレクト信号
DMA / $\overline{\text{DMA}}$	DMA転送期間を示す信号
$\overline{\text{DBSL}}$	外部データバスが選ばれている期間を示す信号（負論理）
S0・S1	CPUのOPコードフェッチサイクルを示す（正論理）
S0+S1	CPUのHALT状態を示す（負論理）
+5V	電源（+5V）ライン
GND	グランドライン
NC	（TK-85では使用されておりません）

12. RAM 増設

TK-85 にはモニタプログラムのワーキングエリアと自作プログラムエリアとして最小のRAM 1kバイトが実装されています．しかし自作プログラムの容量が増えるにつれて，RAM を増設したくなるでしょう．そのため TK-85 では 74LS139 より出力されるアドレスデコード信号をカードエッジに接続しており，外部でこの信号を利用して簡単に 3 kバイト RAM 増設ができるように考慮されています．

カードエッジより出力される $\overline{\text{CS1}}$，$\overline{\text{CS2}}$，$\overline{\text{CS3}}$ はそれぞれ次のように 1 kバイトごとにアドレスデコードしたものです．

$\overline{\text{CS1}}$：8400H～87FFH

$\overline{\text{CS2}}$：8800H～8BFFH

$\overline{\text{CS3}}$：8C00H～8FFFH

増設は 1 kバイトごとに可能です．図11.14はμPD2114LC を用いた増設回路例です．この場合外部に増設するため TK-85 にオプションのμPB8216 を実装して下さい．

また，外部に ROM，RAM を数多く増設する場合，アドレスバス，データバスをバッファを用いて強化してから増設を行ってください．

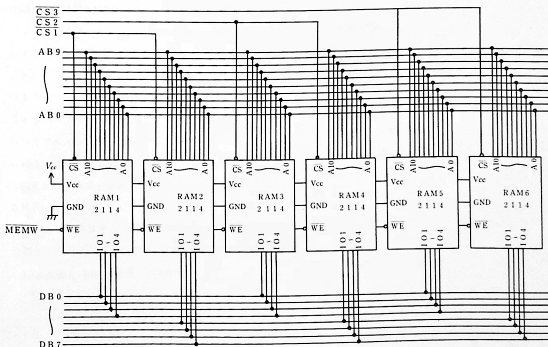


図11.14 RAM 増設例

付　録

1．μPD8085ACデータシート

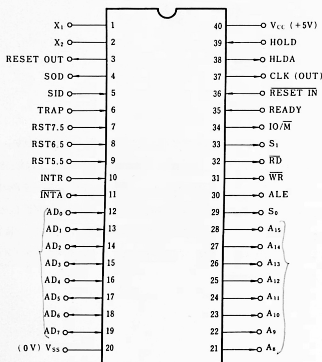
μPD8085Aは，μPD8080AFとソフトウェア・コンパチブルな8 BIT CPUで，μPD8080AFシステムよりシステムの高速性を増し，3チップ：μPD8085A(CPU)，μPD8155／8156(RAM+I／O)，μPD8355／8755A*(ROM／PROM+I／O）で最小システムを構成できます。μPD8085Aは，μPD8080AFに必要なμPB8224（クロック・ジェネレータ)，μPB8228（システム・コントローラ）の総ての機能を含み，システムの集積度を上げています。

μPD8085Aは，マルチプレクス方式のデータバスを採用しており，アドレスの下位8ビットは8 BITのアドレス／データ・バスより出力されますが，μPD8155／8156／8355／8755Aメモリのオンチップ・アドレス・ラッチにより，μPD8085Aと直接インタフェース可能です。

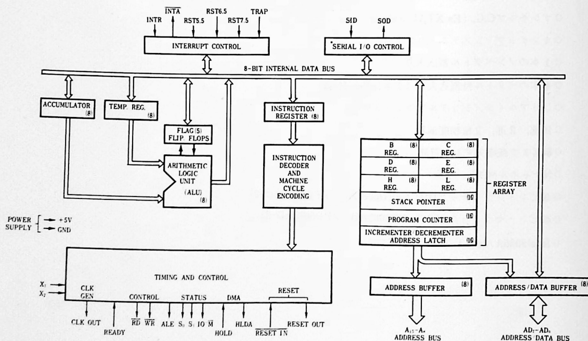
特　徴

○μCOM-85
○単電源　+5V
○μPD8080AFと100%ソフトウェア・コンパチブル
○インストラクション・サイクル1.3μs（μPD8085A）；0.8μs（μPD8085A-2)
○オンチップC.G.（Ex XTAL or RC)
○オンチップ・システム・コントローラ
○1本のノンベクトル割込入力
○4本のベクトル付割込入力（1本はNon Mask)
○シリアルイン／シリアルアウト・ポート
○10進，2進，2倍精度演算
○64Kまで直接アドレス可能
○NチャネルMOS
○40ピン・プラスチックDIP（μPD8085AC，μPD8085AC-2)
○40ピン・セラミックDIP（μPD8085AD，μPD8085AD-2)
○Intel8085A/8085A-2コンパチブル

端子接続図（Top View）



ブロック図



端子機能

（1） A_8-A_{15}（Address Bus）……3ステート出力

メモリアドレスの上位8ビットまたはI/O用8ビットアドレスとなります。ホールドとホールトモード中はハイ・インピーダンスとなります。

（2） AD_0-AD_7（Multiplexed Address/Data Bus）……3ステート入出力

最初のクロックサイクルでメモリアドレスの下位8ビット（またはI/Oアドレス）を出力します。2，3番目のクロックサイクルでは双方向の8ビット・データバスとなります。

ホールドとホールトモード中はハイ・インピーダンスとなります。

（3） ALE（Address Latch Enable）……出力

最初のクロックで発生し，AD_{0-7}のアドレス信号を周辺チップ内のラッチにラッチ入力するために用いられます。ALEの立下りエッジでアドレス情報のセット・ホールド時間が規定されます。

ALEはステータス情報のストローブ信号としても用いられます。

（4） S_0, S_1（Data Bus Status）……出力

次に示すバスサイクルのエンコードされたステータスが出力されます。

S_1	S_0	
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

S_1はアドバンストR/$\overline{W}$ステータスとしても用いられます。

（5） $\overline{RD}$（Read）……3ステート出力

選択されたメモリまたはI/Oが読出され，データバスがデータ転送のために使用できることを示します。

ホールドとホールトモード中はハイ・インピーダンスとなります。

（6） $\overline{WR}$（Write）……3ステート出力

データバス上のデータを選択されたメモリまたはI/Oに書込むために用いられます。

データのセット時間は$\overline{WR}$の後縁より規定されます。

ホールドとホールトモード中はハイ・インピーダンスとなります。

（7） READY……入力

リードまたはライトサイクルにおいてREADY入力がハイならば，メモリまたは周辺チップがデータの送出または受信の準備ができていることを示します。

もしREADY入力がロウならば，CPUはリードまたはライトサイクルを延長し，READYがハイになるのを待ちます。

（8） HOLD……入力

別のマスタがアドレスとデータバスの使用を要求していることを示します。CPUはホールド要求を受けたら，そのときのマシンサイクル終了後，ただちにバスの使用をやめます。ただし内部処理は続けられます。CPUはホールド要求がなくなって始めてバスを使用できるようになります。ホールド要求が受付けられると，アドレス，データ，$\overline{RD}$，$\overline{WR}$，そしてIO/$\overline{M}$の各ラインがハイ・インピーダンスとなります。

（9） HLDA（Hold Acknowledge）……出力

CPUがホールド要求を受付け，次のクロックサイクルからバスの使用をやめることを示します。HLDAはホールド要求がなくなった後ロウになります。

CPUはHLDAがロウになった½クロックサイクル後にバスの使用を開始します。

（10） INTR（Interrupt Reguest）……入力

汎用割込入力として用いられます。INTRは命令の最後から２番目のクロックサイクルでのみ検出されます。もしアクティブであれば，PCのインクリメントは禁止され，$\overline{\text{INTA}}$が発生されます。

割込サービスルーチンにジャンプさせるため，このサイクル間にRESTARTまたはCALLを挿入することができます。

INTRはソフトウェアで許可，禁止を制御できます。さらに外部リセットとこの割込が受付けられた直後に禁止されます。

（11） $\overline{\text{INTA}}$（Interrupt Acknowledge）……出力

この信号はINTRが受付けられた後の命令サイクルで$\overline{\text{RD}}$の代りに（そして$\overline{\text{RD}}$と同じタイミングで）用いられます。これは8259割込チップまたは他の割込ポートをアクティブにするために用いることができます。

（12） RST 5.5, RST 6.5, RST 7.5（Restart Interrupts）……入力

これらの３つの入力は自動的に挿入される内部RESTARTを起こすことを除けば，INTRと同じタイミングを持っています。

これらの割込みの優先順位を次に示します。またこれらの割込みの優先順位はINTRより高くなっています。

RST 7.5	最高優先
RST 6.5	↑
RST 5.5	最低優先

（13） TRAP（Trap Interrupt）……入力

トラップ割込みはノンマスカブル・リスタート割込みです。

これはINTRと同じタイミングで検出されます。この割込みは，いかなるマスクまたは割込イネーブル制御によっても影響を受けません。この割込みは他のどの割込みよりも優先順位が高くなっています。

（14） $\overline{\text{RESET IN}}$……入力

リセットによりPCがゼロにされ，割込イネーブルとHLDAフリップフロップがリセットされます。他のフラグやレジスタ（インストラクション・レジスタを除く）はリセットの影響を受けません。リセットが入力されている間，CPUはリセット状態を続けます。リセットが解除されると０番地からプログラムをスタートします。

（15） RESET OUT……出力

CPUがリセット中であることを示す信号で，システムリセット信号として用いることができます。

この信号はCPUクロックに同期化されています。

（16） X_1，X_2……入力

内部クロック発生回路用のクリスタルまたはRC回路を接続します。さらにX_1は外部クロック入力端子として用いられます。

(17) CLK (Clock) ……出力

クリスタルまたはCR回路によってCPUクロックを発生させている場合に，システムクロック用のクロック出力として用います。

(18) IO/$\overline{M}$ (IO/Memory) ……3ステート出力

この信号がハイならばリード／ライトがI/Oに対して行われ，ロウならばメモリに対して行われることを示します。

ホールドとホールトモード中はハイ・インピーダンスとなります。

(19) SID (Serial Input Data) ……入力

RIM命令が実行されたとき，このラインのデータがアキュムレータのビット7にロードされます。シリアルデータのインタフェースに使用されます。

(20) SOD (Serial Output Data) ……出力

この出力はSIM命令によってセット，リセットされます。

シリアルデータのインタフェースに使用されます。

割込みとシリアルI/O

8085Aは5本の割込入力(INTR，RST 5.5，RST 6.5，RST 7.5そしてTRAP)を持っています。INTRは8080AのINTと同一機能になっています。RST 5.5，6.5，7.5およびTRAPはリスタート割込みですが，TRAPはノンマスク，他の3本はプログラムでマスク可能です。

RST 5.5，6.5，7.5の割込入力がイネーブルになったとき，インタラプト・マスクがセットされていなければ，PCの内容をスタックに退避させ，リスタートアドレスに分岐します。TRAP割込みがアクティブになると，インタラプト・イネーブルやマスクに関係なくリスタートアドレスに分岐します。

割込名称	リスタートアドレス (16進)
TRAP	24
RST 5.5	2C
RST 6.5	34
RST 7.5	3C

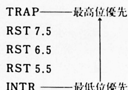
RST 5.5とRST 6.5はINTR(そして8080AのINT)と同様に，ハイレベル検出になっており，INTRと同じタイミングで受付けられます。RST 7.5は立上りエッジ検出になっています。

RST 7.5に対しては，内部割込要求を発生させるための内部F/Fをセットするのに1個のパルスだけで十分です。RST 7.5リクエストF/Fは，そのリクエストが処理されるまでセットし続け，処理された後自動的にリセットされます。このF/FはSIM命令または外部リセット入力 ($\overline{\text{RESET IN}}$) によってもリセットされます。

なおRST 7.5内部F/Fは，RST 7.5がマスクされているときでもRST 7.5端子に入力された1個のパルスによってセットされます。

3個のRSTインタラプト・マスクはSIM命令または$\overline{\text{RESET IN}}$によってのみ影響されます。

1個以上の割込みが同時に起こったとき，どれを受付けるかを決定するために，各割込みに対して次に示す優先順位が定めてあります。

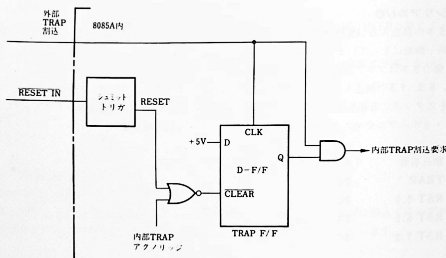


この優先機構は，より高い優先の割込みによってスタートされていたルーチンの優先順位は考慮していません。例えば，RST 5.5は，RST 7.5ルーチンの終了前に各割込みがイネーブルにされていれば，RST 7.5ルーチンに割込むことができます。

TRAP割込みは，電源故障やバスエラーなどに有効です。TRAP割込みは他の割込みと同様に受付けられますが，優先順位は最も高くなっています。TRAP割込みは，どのフラグ，マスクによっても影響されません。TRAP入力はエッジおよびレベルの両方で検出されます。

TRAP入力が受付けられるためには，ハイに立上り，その後ハイを保持しなければなりませんが，一度ロウに下がってそれからハイに上がるまでは，再び受付けられることはありません。これによりノイズやロジックグリッチによる誤トリガを避けることができます。

次図に8085A内のTRAP割込要求回路を示します。



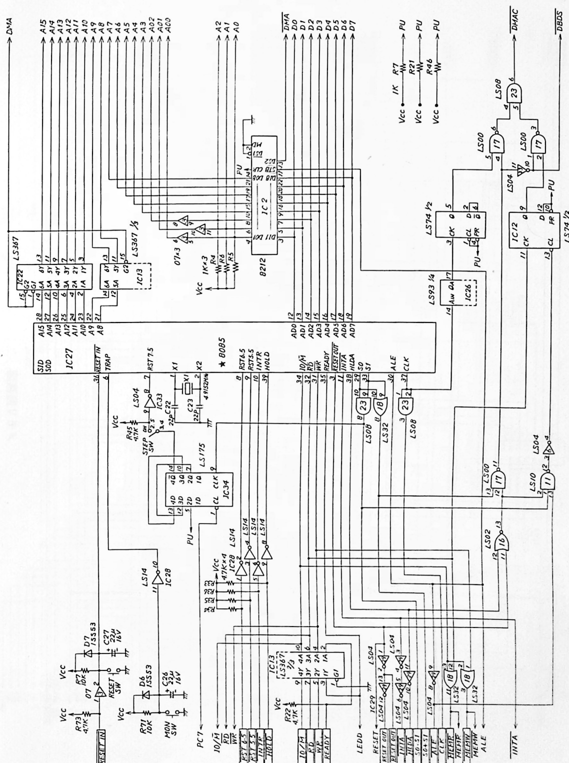
どの割込み（TRAP，RST 7.5，RST 6.5，RST 5.5，INTR）でも一度その割込処理に入ると，EI命令が実行されるまではその後に入力されるすべての割込み（TRAPを除いて）を禁止するということに注意して下さい。

TRAP割込みは，他の割込みがイネーブル状態でもそうでなくても割込み発生し，他の割込みを禁止させることができます。

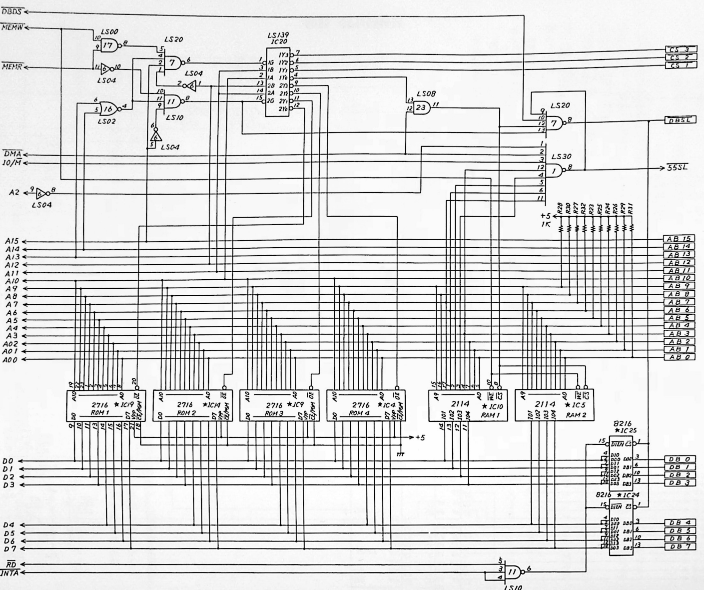
TRAP割込み処理に入った後，RIM命令を実行することによって以前のインタラプト・イネーブル・フラグの状態を知ることができます。

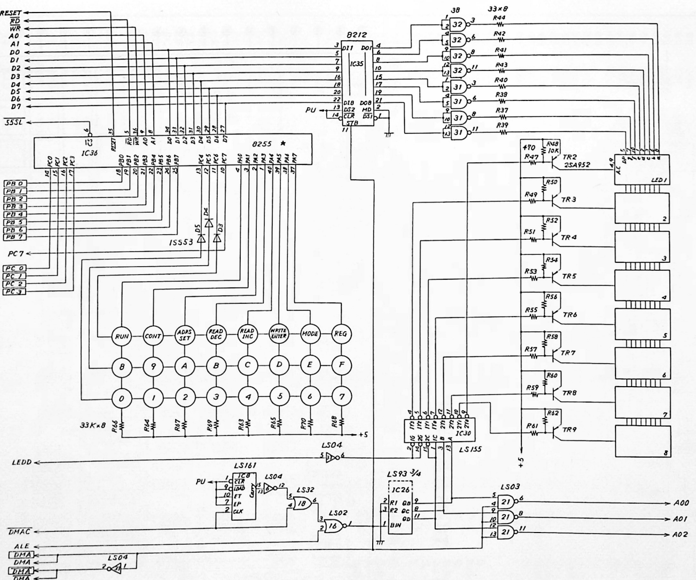
シリアルI/OシステムもまたRIMとSIM命令によって制御されます。SIDはRIM命令によって読まれ，SODデータはSIM命令によってセットされます。

2.　TK-85全回路図

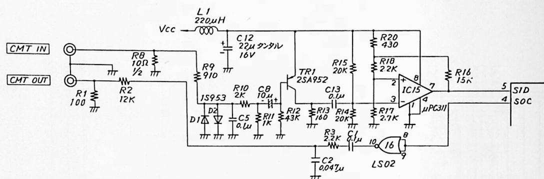


8085 CPU回路図


CS 3
CS 2
CS 1
DBSL
55SL
AB 15
AB 14
AB 13
AB 12
AB 11
AB 10
AB 9
AB 8
AB 7
AB 6
AB 5
AB 4
AB 3
AB 2
AB 1
AB 0
DB 0
DB 1
DB 2
DB 3
DB 4
DB 5
DB 6
DB 7
8216 ★IC25
8216 ★IC24
2114 ★IC5 RAM 2
2114 ★IC10 RAM 1
2716 ★IC4 ROM 4
2716 ★IC9 ROM 3
2716 ★IC14 ROM 2
2716 ★IC19 ROM 1
LS139 IC20
LS20
LS30
LS08
LS10
LS04
LS00
LS02
+5
1K
DBDS
MEMW
MEMR
DMA
IO/M
A2
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A02
A01
A00
D0
D1
D2
D3
D4
D5
D6
D7
RD
INTA
メモリ回路図

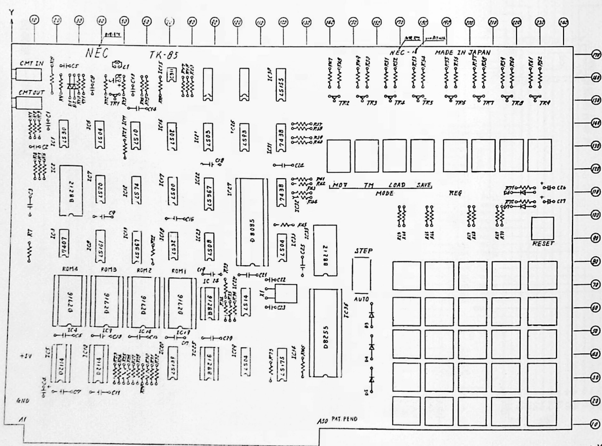


LED，キーボード回路図



CMT回路図

3．　TK-85部品配置図



μCOM-85　インストラクション活用表

μCOM-85　機械語↔ニーモニック　対照表

00	NOP		40	MOV	B, B	80	ADD	B	C0	RNZ	
01	LXI	B, B_3B_2	41	MOV	B, C	81	ADD	C	C1	POP	B
02	STAX	B	42	MOV	B, D	82	ADD	D	C2	JNZ	B_3B_2
03	INX	B	43	MOV	B, E	83	ADD	E	C3	JMP	B_3B_2
04	INR	B	44	MOV	B, H	84	ADD	H	C4	CNZ	B_3B_2
05	DCR	B	45	MOV	B, L	85	ADD	L	C5	PUSH	B
06	MVI	B, B_2	46	MOV	B, M	86	ADD	M	C6	ADI	B_2
07	RLC		47	MOV	B, A	87	ADD	A	C7	RST	0
08	・・・		48	MOV	C, B	88	ADC	B	C8	RZ	
09	DAD	B	49	MOV	C, C	89	ADC	C	C9	RET	
0A	LDAX	B	4A	MOV	C, D	8A	ADC	D	CA	JZ	B_3B_2
0B	DCX	B	4B	MOV	C, E	8B	ADC	E	CB	・・・	
0C	INR	C	4C	MOV	C, H	8C	ADC	H	CC	CZ	B_3B_2
0D	DCR	C	4D	MOV	C, L	8D	ADC	L	CD	CALL	B_3B_2
0E	MVI	C, B_2	4E	MOV	C, M	8E	ADC	M	CE	ACI	B_2
0F	RRC		4F	MOV	C, A	8F	ADC	A	CF	RST	1
10	・・・		50	MOV	D, B	90	SUB	B	D0	RNC	
11	LXI	D, B_3B_2	51	MOV	D, C	91	SUB	C	D1	POP	D
12	STAX	D	52	MOV	D, D	92	SUB	D	D2	JNC	B_3B_2
13	INX	D	53	MOV	D, E	93	SUB	E	D3	OUT	B_2
14	INR	D	54	MOV	D, H	94	SUB	H	D4	CNC	B_3B_2
15	DCR	D	55	MOV	D, L	95	SUB	L	D5	PUSH	D
16	MVI	D, B_2	56	MOV	D, M	96	SUB	M	D6	SUI	B_2
17	RAL		57	MOV	D, A	97	SUB	A	D7	RST	2
18	・・・		58	MOV	E, B	98	SBB	B	D8	RC	
19	DAD	D	59	MOV	E, C	99	SBB	C	D9	・・・	
1A	LDAX	D	5A	MOV	E, D	9A	SBB	D	DA	JC	B_3B_2
1B	DCX	D	5B	MOV	E, E	9B	SBB	E	DB	IN	B_2
1C	INR	E	5C	MOV	E, H	9C	SBB	H	DC	CC	B_3B_2
1D	DCR	E	5D	MOV	E, L	9D	SBB	L	DD	・・・	
1E	MVI	E, B_2	5E	MOV	E, M	9E	SBB	M	DE	SBI	B_2
1F	RAR		5F	MOV	E, A	9F	SBB	A	DF	RST	3
20	RIM		60	MOV	H, B	A0	ANA	B	E0	RPO	
21	LXI	H, B_3B_2	61	MOV	H, C	A1	ANA	C	E1	POP	H
22	SHLD	B_3B_2	62	MOV	H, D	A2	ANA	D	E2	JPO	B_3B_2
23	INX	H	63	MOV	H, E	A3	ANA	E	E3	XTHL	
24	INR	H	64	MOV	H, H	A4	ANA	H	E4	CPO	B_3B_2
25	DCR	H	65	MOV	H, L	A5	ANA	L	E5	PUSH	H
26	MVI	H, B_2	66	MOV	H, M	A6	ANA	M	E6	ANI	B_2
27	DAA		67	MOV	H, A	A7	ANA	A	E7	RST	4
28	・・・		68	MOV	L, B	A8	XRA	B	E8	RPE	
29	DAD	H	69	MOV	L, C	A9	XRA	C	E9	PCHL	
2A	LHLD	B_3B_2	6A	MOV	L, D	AA	XRA	D	EA	JPE	B_3B_2
2B	DCX	H	6B	MOV	L, E	AB	XRA	E	EB	XCHG	
2C	INR	L	6C	MOV	L, H	AC	XRA	H	EC	CPE	B_3B_2
2D	DCR	L	6D	MOV	L, L	AD	XRA	L	ED	・・・	
2E	MVI	L, B_2	6E	MOV	L, M	AE	XRA	M	EE	XRI	B_2
2F	CMA		6F	MOV	L, A	AF	XRA	A	EF	RST	5
30	SIM		70	MOV	M, B	B0	ORA	B	F0	RP	
31	LXI	SP, B_3B_2	71	MOV	M, C	B1	ORA	C	F1	POP	PSW
32	STA	B_3B_2	72	MOV	M, D	B2	ORA	D	F2	JP	B_3B_2
33	INX	SP	73	MOV	M, E	B3	ORA	E	F3	DI	
34	INR	M	74	MOV	M, H	B4	ORA	H	F4	CP	B_3B_2
35	DCR	M	75	MOV	M, L	B5	ORA	L	F5	PUSH	PSW
36	MVI	M, B_2	76	HLT		B6	ORA	M	F6	ORI	B_2
37	STC		77	MOV	M, A	B7	ORA	A	F7	RST	6
38	・・・		78	MOV	A, B	B8	CMP	B	F8	RM	
39	DAD	SP	79	MOV	A, C	B9	CMP	C	F9	SPHL	
3A	LDA	B_3B_2	7A	MOV	A, D	BA	CMP	D	FA	JM	B_3B_2
3B	DCX	SP	7B	MOV	A, E	BB	CMP	E	FB	EI	
3C	INR	A	7C	MOV	A, H	BC	CMP	H	FC	CM	B_3B_2
3D	DCR	A	7D	MOV	A, L	BD	CMP	L	FD	・・・	
3E	MVI	A, B_2	7E	MOV	A, M	BE	CMP	M	FE	CPI	B_2
3F	CMC		7F	MOV	A, A	BF	CMP	A	FF	RST	7

μCOM-85　ニーモニック↔機械語　対照表

(MOV, INR, DCR, MVI) 命令

r_2(r)	A	B	C	D	E	H	L	M
MOV A, r_2	7F	78	79	7A	7B	7C	7D	7E
MOV B, r_2	47	40	41	42	43	44	45	46
MOV C, r_2	4F	48	49	4A	4B	4C	4D	4E
MOV D, r_2	57	50	51	52	53	54	55	56
MOV E, r_2	5F	58	59	5A	5B	5C	5D	5E
MOV H, r_2	67	60	61	62	63	64	65	66
MOV L, r_2	6F	68	69	6A	6B	6C	6D	6E
MOV M, r	77	70	71	72	73	74	75	—
INR r	3C	04	0C	14	1C	24	2C	34
DCR r	3D	05	0D	15	1D	25	2D	35
MVI r, B_2	3E	06	0E	16	1E	26	2E	36

(LXI, DAD, INX, DCX) 命令

×	B	D	H	SP
LXI ×, B_3B_2	01	11	21	31
DAD ×	09	19	29	39
INX ×	03	13	23	33
DCX ×	0B	1B	2B	3B

スタック操作 命令

×	B	D	H	PSW
PUSH ×	C5	D5	E5	F5
POP ×	C1	D1	E1	F1

演算・論理 (I)

r	A	B	C	D	E	H	L	M
ADD r	87	80	81	82	83	84	85	86
ADC r	8F	88	89	8A	8B	8C	8D	8E
SUB r	97	90	91	92	93	94	95	96
SBB r	9F	98	99	9A	9B	9C	9D	9E
ANA r	A7	A0	A1	A2	A3	A4	A5	A6
XRA r	AF	A8	A9	AA	AB	AC	AD	AE
ORA r	B7	B0	B1	B2	B3	B4	B5	B6
CMP r	BF	B8	B9	BA	BB	BC	BD	BE

演算・論理 (II)

ADI B_2	C6
ACI B_2	CE
SUI B_2	D6
SBI B_2	DE
ANI B_2	E6
XRI B_2	EE
ORI B_2	F6
CPI B_2	FE

Accとのデータ転送(8ビット)

	STAX B	LDAX B	STAX D	LDAX D	STA B_3B_2	LDA B_3B_2
オペレーション	[(B)(C)]⇆(A)		[(D)(E)]⇆(A)		[⟨B_3⟩⟨B_2⟩]⇆(A)	
機械語	02	0A	12	1A	32	3A

回転命令

	RLC	RRC	RAL	RAR
オペレーション	$A_0 \leftarrow A_7$	$A_0 \rightarrow A_7$	$A_0 \leftarrow C$	$C \rightarrow A_7$
機械語	07	0F	17	1F

HLレジスタとのデータ転送(16ビット)

	SHLD B_3B_2	LHLD B_3B_2	XTHL	XCHG	PCHL	SPHL
オペレーション	[⟨B_3⟩⟨B_2⟩+1] [⟨B_3⟩⟨B_2⟩] ↑ (H)(L)	[⟨B_3⟩⟨B_2⟩+1] [⟨B_3⟩⟨B_2⟩] ↓ 〃	[(SP)+1] [(SP)] ↕ 〃	(D)(E) ↕ 〃	(PC) ↑ 〃	(SP) ↑ 〃
機械語	22	2A	E3	EB	E9	F9

入出力・割込制御命令

	OUT B_2	IN B_2	DI	EI
機械語	D3	DB	F3	FB

Acc補正・C操作命令

	DAA	CMA	STC	CMC
オペレーション	補正	$A \leftarrow \bar{A}$	$C \leftarrow 1$	$C \leftarrow \bar{C}$
機械語	27	2F	37	3F

ジャンプ, コール, リターン 命令

	---→	NZ	Z	NC	C	PO	PE	P	M
JMP B_3B_2	C3	C2	CA	D2	DA	E2	EA	F2	FA
CALL B_3B_2	CD	C4	CC	D4	DC	E4	EC	F4	FC
RET	C9	C0	C8	D0	D8	E0	E8	F0	F8
条件	---→	Z=0,	Z=1	C=0,	C=1	P=0,	P=1	S=0,	S=1

その他の命令

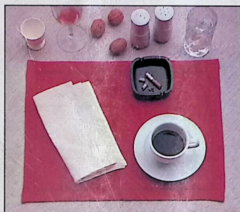
	HLT	NOP
機械語	76	00

リスタート 命令

×	0	1	2	3	4	5	6	7
RST ×	C7	CF	D7	DF	E7	EF	F7	FF

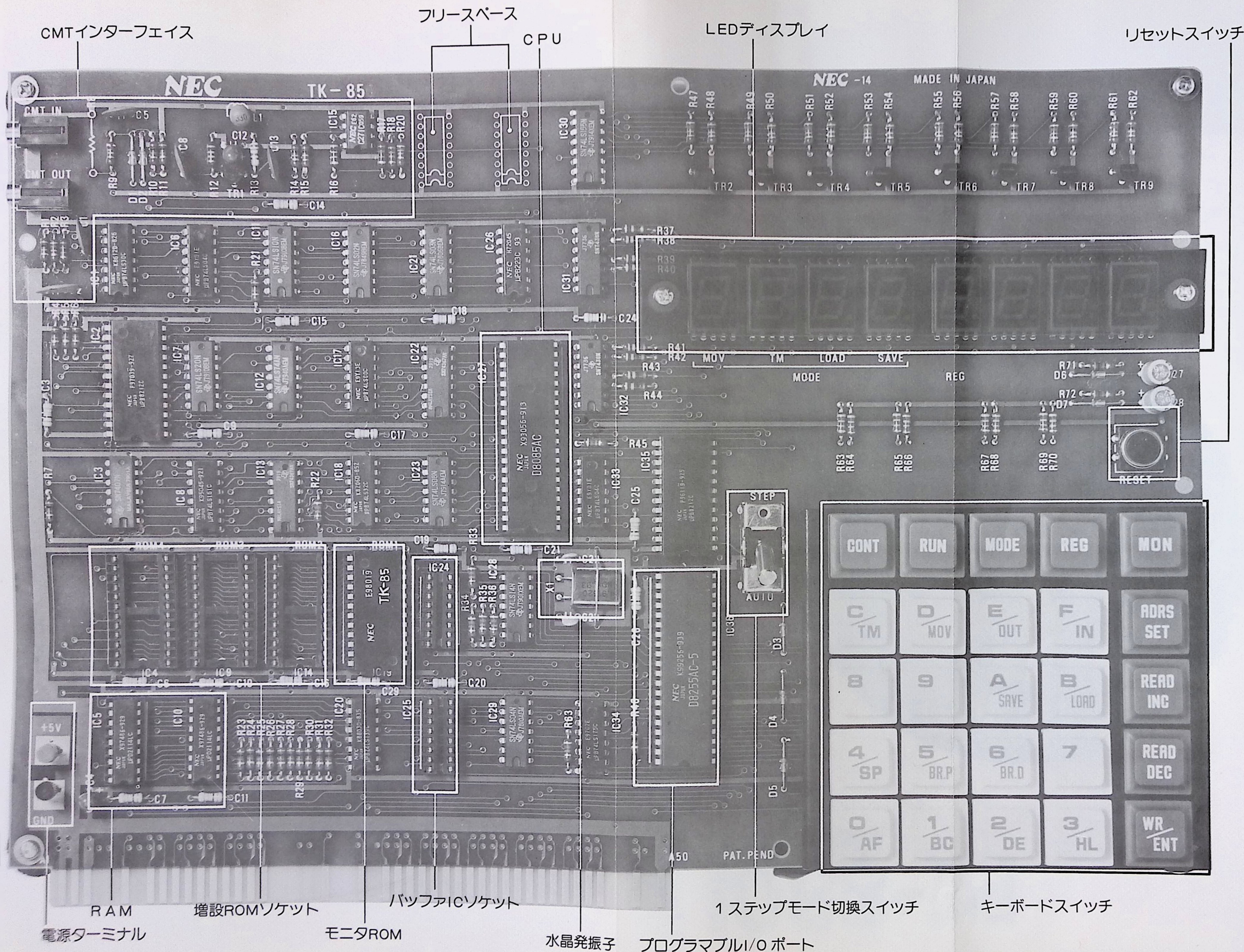
85で追加された命令

	RIM	SIM
機械語	20	30



Something is Happening...

NEC 日本電気株式会社



〔本写真の使用部品は改良のため予告なく変更されることがあります〕